



AFRL-RQ-WP-TR-2016-0001

RUNTIME ASSURANCE FRAMEWORK DEVELOPMENT FOR HIGHLY ADAPTIVE FLIGHT CONTROL SYSTEMS

**John D. Schierman, Michael D. DeVore, Nathan D. Richards, Neha Gandhi, Jared K. Cooper,
and Kenneth R. Horneman**

Barron Associates, Inc.

Scott Stoller and Scott Smolka

Stony Brook University

DECEMBER 2015

Final Report

THIS IS A SMALL BUSINESS INNOVATION RESEARCH (SBIR) PHASE III REPORT.

Approved for public release; distribution unlimited.

See additional restrictions described on inside pages

**AIR FORCE RESEARCH LABORATORY
AEROSPACE SYSTEMS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7541
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the USAF 88th Air Base Wing (88 ABW) Public Affairs Office (PAO) and is available to the general public, including foreign nationals.

Copies may be obtained from the Defense Technical Information Center (DTIC)
(<http://www.dtic.mil>).

AFRL-RQ-WP-TR-2016-0001 HAS BEEN REVIEWED AND IS APPROVED FOR
PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

*//Signature//

MATTHEW A. CLARK, Chief
Autonomous Control Branch
Power and Control Division

//Signature//

MICHAEL W. OPPENHEIMER
Technical Advisor
Autonomous Control Branch
Power and Control Division

This report is published in the interest of scientific and technical information exchange and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

*Disseminated copies will show “//Signature//” stamped or typed above the signature blocks.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
<p>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</p>					
1. REPORT DATE (DD-MM-YY) December 2015		2. REPORT TYPE Final		3. DATES COVERED (From - To) 27 September 2012 – 30 November 2015	
4. TITLE AND SUBTITLE RUNTIME ASSURANCE FRAMEWORK DEVELOPMENT FOR HIGHLY ADAPTIVE FLIGHT CONTROL SYSTEMS				5a. CONTRACT NUMBER FA8650-12-C-3227	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER 62201F	
6. AUTHOR(S) John D. Schierman, Michael D. DeVore, Nathan D. Richards, Neha Gandhi, Jared K. Cooper, and Kenneth R. Horneman (Barron Associates, Inc.) Scott Stoller and Scott Smolka (Stony Brook University)				5d. PROJECT NUMBER 2403	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER Q0PA	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Barron Associates, Inc. 1410 Sachem Place, Suite 202 Charlottesville, VA 22901				8. PERFORMING ORGANIZATION REPORT NUMBER FTR_469	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory Aerospace Systems Directorate Wright-Patterson Air Force Base, OH 45433-7541 Air Force Materiel Command United States Air Force				10. SPONSORING/MONITORING AGENCY ACRONYM(S) AFRL/RQQA	
				11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S) AFRL-RQ-WP-TR-2016-0001	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.					
13. SUPPLEMENTARY NOTES This is a Small Business Innovation Research (SBIR) Phase III report. Barron Associates, Inc. has waived its SBIR data rights, and the report has been approved for public release (PA Case Number: 88ABW-2016-1452; Clearance Date: 11 March 2016).					
14. ABSTRACT This report was developed under a SBIR contract. This report describes the technical progress made by Barron Associates, Inc. and its partners in runtime assurance (RTA) systems, which hold the promise of protecting advanced systems that cannot be fully certified at design time due to their inherent complexity. A number of technical hurdles remain in the implementation of RTA systems for highly complex safety-critical systems, and the main objective of this effort was to further address these issues. One main focus of this project was to investigate the necessary structure of RTA frameworks for multi-level interacting feedback systems. As such, a challenge problem was constructed for a fleet of unmanned aircraft systems (UASs) performing a surveillance mission. The demonstration platform consisted of RTA systems for the inner-loop control, outer-loop guidance, ownship flight management, and fleet mission planning elements. The framework design and certification requirements for such a system were explored in this program. For the inner-loop, the concept of employing multiple transition controllers in the reversionary control system was studied. For all feedback levels, the required RTA checks were developed and the critical reversionary switching conditions defined. The interactions between the RTA protected systems and certified collision avoidance systems were also investigated. A safety case argument for design-time certification of the RTA protected systems was constructed using subsystem requirements contracts that were developed from a compositional reasoning approach explored over the course of the project.					
15. SUBJECT TERMS SBIR Report, Runtime assurance, formal methods, V&V, certification, unmanned aircraft systems					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT: SAR	18. NUMBER OF PAGES 310	19a. NAME OF RESPONSIBLE PERSON (Monitor) Matthew A. Clark 19b. TELEPHONE NUMBER (Include Area Code) N/A
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified			



1410 SACHEM PLACE
SUITE 202
CHARLOTTESVILLE, VA 22901

P 434.973.1215
F 434.973.4686
W BARRON-ASSOCIATES.COM
E BARRON@BAINET.COM

June 7, 2016

Matthew Clark
Chief, Autonomous Control Branch
Air Force Research Laboratory AFRL/RQQA
2210 Eighth St, Room 301
Wright-Patterson AFB, OH 45433

Subject: Contract No. FA8650-12-C-3227, SBIR Phase III

Dear Mr. Clark:

Barron Associates, Inc. hereby waives its SBIR Data Rights to all contents of the final report for the subject contract. The Government is granted an unlimited nonexclusive license to use, modify, reproduce, release, perform, and display or disclose this report and the data contained herein.

We affirm that we are aware that the report may be released to other contractors to the Government and approve potential release to other contractors.

Sincerely,

John D. Schierman
Principal Research Scientist
BARRON ASSOCIATES, INC.
1410 Sachem Place, Suite 202
Charlottesville, VA 22901

Table of Contents

Section	Page
List of Figures	v
List of Tables	viii
Acknowledgements	ix
1 Executive Summary	1
2 Introduction	6
2.1 Overview of Program and Report	6
2.2 Motivation and Background	6
2.3 Definition of RTA and the RTA Protected System	8
2.3.1 RTA Protected System Framework	9
2.4 Current V&V and Certification Processes	10
2.4.1 Review of Industry Standards and Regulations	10
2.4.2 Runtime Assurance for System Safety	16
2.4.3 RTA Safety Case Argument	18
2.5 Program Objectives, Scope, and Challenge Problem	19
2.6 Technical Approach	22
2.6.1 Key Assumption on New Approach	22
2.6.2 Generalizing the RTA Architecture	22
2.7 Review of Technical Challenges in RTA System Development	23
2.7.1 Performance Limitations	23
2.7.2 Safety Boundary Construction	23
2.7.3 Switching Condition Determination	25
2.7.4 Vehicle Health and Contingency Management	26
2.7.5 Design-Time V&V Methods for the RTA System	28
3 Fundamentals of the Developed RTA Framework	29
3.1 RTA Protected System	29
3.1.1 General Description of Elements within the RTA Protected Framework	29
3.2 Formal Definitions of Safety Levels	30
3.2.1 General Concept of Safety	31
3.2.2 Determined Safety (Dsafe Space)	31
3.2.3 Safety Levels for Runtime Checking	33
3.2.4 Illustrative Example	37
3.3 Reversionary System Description	39
3.3.1 Reversionary System Composition	39
3.3.2 Illustrative Example Continued	41
3.3.3 Reversionary System Shadow Mode Operations	42
3.4 Interacting RTA Protected Systems	42
3.4.1 RTA Monitor Checks	43
3.4.2 RTA Monitor Viewed as an Assume-Guarantee Contract Checker	45
3.4.3 The Need for a Global RTA Manager	46
3.5 Additional RTA System Considerations	49
3.5.1 Switching Protocol	49
3.5.2 Complex versus Simple Boundary Checks	49
3.5.3 RTA Frameworks Operating with Other System Code	50

Table of Contents (Continued)

Section	Page
3.5.4 Certifiable Code Residing within Advanced Systems	51
4 RTA Applied to the Selected Challenge Problem	52
4.1 Challenge Problem Selection Objectives	52
4.2 Overview of Selected Challenge Problem	52
4.3 Feedback System for Challenge Problem Ownship UAS	54
4.4 Expected Approach for Introducing RTA Protected Systems	56
4.5 A Modular RTA Protected System Framework	59
4.5.1 Motivation	59
4.5.2 Input/Output Protocol for a Modular RTA Protected Framework	59
5 RTA Protection Applied to Inner-Loop Control Systems	61
5.1 General Description of Morphing Wing Vehicle	61
5.2 Reversionary CLAW Description	62
5.2.1 Transitions Controller Designs	62
5.3 Type Safety at Inner-Loop Control Law Level	64
5.4 A-G Contracts at Inner-Loop Control Law Level	65
5.4.1 Assumptions on Inputs from Upstream Guidance System	65
5.4.2 A-G Contracts for Control Effectors	66
5.4.3 A-G Contracts for Sensors	67
5.4.4 A-G Contracts for the Airframe/Engine	67
5.4.5 A-G Contracts for the Inner-Loop Controller	68
5.4.6 A-G Contracts for the CLAW RTA System	69
5.5 Inner-Loop RTA Checks and Switching Conditions	70
5.6 Inner-Loop Experimental Results	70
5.6.1 RTA Demonstration: Mismanaged Wing Morphing	71
5.6.2 RTA Demonstration: Error in Advanced Controller	76
6 RTA Protection Applied to Outer-Loop Guidance Systems	81
6.1 General Description of 3-DOF UAS Model used at the Guidance Level	81
6.2 GLAW Functionality	81
6.2.1 Difference between Advanced and Reversionary GLAWs	82
6.3 Type Safety at Outer-Loop GLAW Level	83
6.4 A-G Contracts at Outer-Loop GLAW Level	87
6.4.1 Assumptions on Inputs from the Upstream FMS	88
6.4.2 A-G Contracts for Certified Closed Inner-Loop System	89
6.4.3 A-G Contracts for the Outer-Loop Guidance System	89
6.4.4 A-G Contracts for the GLAW RTA System	89
6.5 Outer-Loop GLAW RTA Checks and Switching Conditions	90
6.6 Experimental Results	91
7 RTA Protection Applied to Flight Management Systems	95
7.1 General Description of the FMS Functionality	95
7.1.1 Difference between Advanced and Reversionary FMS Blocks	97
7.2 Safety at the FMS Level	98
7.3 Type Safety at the FMS Level	99
7.3.1 FMS Planning Horizon	101

Table of Contents (Continued)

Section	Page
7.4 A-G Contracts at FMS Level	102
7.4.1 Assumptions on Inputs from the Upstream MPS	103
7.4.2 A-G Contracts for Closed-Loop CLAW/GLAW System	103
7.4.3 A-G Contracts for the FMS	104
7.4.4 A-G Contract for FMS RTA System	105
7.5 FMS RTA Checks and Switching Conditions	105
7.6 Experimental Results	106
7.6.1 FMS RTA Check on Safe Path Existence and Fuel Reserves	106
7.6.3 FMS RTA Check on RSV Separation through Bottleneck	118
8 Certified Collision Avoidance Integrated with RTA Protection	119
8.1 The Need for Certified Collision Avoidance	119
8.2 Recommended Frameworks	120
8.2.1 Certified Collision Avoidance as an Integral Part of RTA	120
8.2.2 Certified Collision Avoidance Separate from RTA	120
8.3 Overview of Collision Avoidance	121
8.3.1 Classes of Collision Avoidance Functions	122
8.3.2 One General Collision Avoidance Algorithm	124
8.4 Certified Reactive Collision Avoidance	125
9 RTA Protection Applied to Mission Planning Systems	130
9.1 Current Mission Planning Protocols	130
9.2 General Description of MPS Functionality	132
9.2.1 Command/Communication Architecture	133
9.2.2 Initial Mission Planning – The Air Tasking Order	133
9.2.3 Difference between Advanced and Reversionary MPS Blocks	136
9.3 Safety at MPS Level	138
9.4 Type Safety at the MPS Level	138
9.5 A-G Contracts at MPS Level	139
9.5.1 Assumptions on the Inputs to the MPS	139
9.5.2 A-G Contracts for Closed-Loop CLAW/GLAW/FMS System	140
9.5.3 A-G Contract for the MPS	140
9.5.4 A-G Contract for the MPS RTA System	141
9.6 MPS RTA Checks and Switching Conditions	141
9.7 Additional Considerations at the Mission Planning Level	142
9.7.1 Building RTA Checks with a Mission Modeling Language	142
9.7.2 Risk/Reward Management	144
9.7.3 Nominal Contingency Planning and the Challenge to RTA Monitoring	145
9.7.4 AFRL Proposed General RTA Modal Architecture	145
9.8 Proposed Experimental Investigations	148
10 Safety Case Argument for an RTA Protected System	151
10.1 General Considerations	151
10.2 GSN Diagrams used to Construct Safety Cases	151
10.3 Safety Case for Challenge Problem using GSN Diagrams	152
11 Alternative Frameworks	163

Table of Contents (Concluded)

Section	Page
11.1 Integrated RTA Protected Feedback Loops	163
11.2 Allowing Post Reversion Restart of Advanced Systems	163
11.3 Diagnostic and Prognostic Checking for Post Mission Analysis	164
12 Conclusions	165
13 Recommendations	168
14 References	170
LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS	177
Appendices	179

List of Figures

Figure	Page
1. The Concept of an RTA Protected System	9
2. RTA Protected Systems in the Overall Framework.....	10
3. Development Assurance Process	12
4. The RTA Protected System and Criticality Level Defined Software	18
5. Nested Feedback Architecture for Challenge Problem.....	20
6. The Generalized Concept of an RTA Protected System.....	29
7. Relationship between the Sets of Actual and Declared Safe States.....	33
8. Relationship between Type I, II, and III Safety Regions.....	36
9. Illustrative Example – Type I, II, and III Safety for Altitude State	38
10. The Role of the Transition and Baseline Systems in the Reversionary System	40
11. Illustrative Example Depicting Transition and Baseline Systems	41
12. Interacting RTA Protected Systems.....	43
13. Global RTA Manager for Multiple Interacting Runtime Protected Systems	46
14. Tracking Performance Reduction due to Overdriving System	48
15. RTA Protected Loops Interacting with Other System Code.....	50
16. Design Time Certified Feedback System for Challenge Problem Ownship UAS.....	54
17. Simplified Version of Feedback System for Challenge Problem Ownship.....	55
18. Build-Up Approach to Introducing RTA Protected Advanced Systems	57
19. Example Combinations of Interacting DTA and RTA Protected Systems.....	58
20. Integrated, Non-Modular RTA Framework (Not Allowed Here).....	60
21. Morphing Wing Baseline Planform Configurations	61
22. Transition Procedure from Advanced to Baseline Control.....	63
23. Development of A-G Contracts at the Inner-Loop Level	66
24. RTA Demonstration: Mismanaged Wing Morphing Procedure.....	72
25. Guarantee of Progress for RTA Demonstration: Mismanaged Morphing.....	73
26. Parameter Time Histories: Mismanaged Morphing with RTA Protection	74
27. Parameter Time Histories: Mismanaged Morphing without RTA Protection	75
28. Parameter Time Histories: Error in Advanced Controller with RTA Protection	77
29. Parameter Time Histories: Error in Advanced Controller without RTA Protection	80
30. Geometry for Standard Waypoint Following Guidance	82
31. The Concept of a Safety Corridor	83
32. Definition of Desired Region Q_G	85
33. Geometric Interpretation of GType I – III Boundaries	86
34. Separation Volumes used in Collision Avoidance Schemes	86
35. Equivalence of GType I – III Boundaries between Safety Corridor and RSVs	87
36. Development of A-G Contracts at the Outer-Loop GLAW Level	88
37. Defined Distances for Calculating GType III Switching Condition.....	92
38. Crosstrack Overshoot due to RGLAW Reversion	92
39. Example Experimental Result for GLAW RTA Protection	94
40. Initial Waypoint Plans Generated by FMSs.....	96
41. Deconflicted Waypoint Plans Generated by FMSs after Negotiation/Iteration.....	97
42. The Bottleneck Problem and FType I, II, and III Boundaries	100
43. FMS Plans Out Path at Future Time	102

List of Figures (Continued)

Figure	Page
44. Development of A-G Contracts at the FMS Level	103
45. Rendezvous Points and Airspeed Schedule for Case Study	107
46. No-Fly Zone Interference with Nominal Mission Path	108
47. 103 Paths Returned by RRT* for the Same Pair of Advanced Waypoints.....	109
48. Case Study Following Short RRT* Computed Path - Mission Completed.....	109
49. Case Study Following Long RRT* Computed Path – Aircraft Runs Out of Fuel.....	110
50. Fuel Remaining during Mission with Long RRT* Avoidance Path.....	110
51. Time-Aloft-Remaining	111
52. Sample Path Home from Current Location	112
53. Shortest Distance and Shortest Time Home along Defined Path	113
54. Estimated Reserve Endurance.....	113
55. Long RRT* Path Case Study - Estimated Reserve with No RTA Protection	114
56. Long RRT* Path Case Study – RTA Reversion Back to Base.....	115
57. Long RRT* Path Case Study – Reserve Endurance and Threshold	115
58. Effects of Forecasted Winds-Aloft on FMS RTA Reserve Endurance Check	117
59. Experimental Results of Required Fleet Separation due to Encountered Bottleneck.....	118
60. Certified Collision Avoidance System as an Integral Part of RTA	120
61. Certified Collision Avoidance System Separate from RTA Framework.....	121
62. Avoidance Trajectory Types: Continuous or Waypoints.....	122
63. Variable Avoidance Distance as a Function of Risk	123
64. No-Fly Zone Margins as a Function of Risk	124
65. Nominal Reactive Collision Avoidance.....	126
66. Predicting Ownship’s Capability to Maintain Required Separation – No Threat	127
67. Predicting Ownship’s Capability to Maintain Separation – Threat Identified	128
68. Incorrect Prediction of Intruder’s Path – Required Separation Not Maintained	129
69. The Battle Rhythm – Reproduced from (Joint Chiefs of Staff, 2014).....	131
70. The Joint Air Tasking Cycle - Reproduced from (Joint Chiefs of Staff, 2014).....	131
71. Example Initial ATO for ISR Mission.....	134
72. The Initial Air Tasking Order Shown Loaded to Each Vehicle’s MPS.....	135
73. Example Certified Initial ATO for the ISR Mission.....	136
74. Two Separate Missions Developed to Accomplish the ATO	137
75. Development of A-G Contracts at the MPS Level	139
76. RTA Protected MPS Consists of Both Design Time and Day-of-Mission Planning Constructs	144
77. AFRL Proposed General RTA Modal Architecture	146
78. AMPS and RMPS Scanning Protocols for the VIP Mission	150
79. Safety Argument Tools Using Goal Structuring Notation.....	152
80. Top Level Safety Argument for Fleet of UASs.....	153
81. Safety Argument Pattern Template for Feedback Level.....	155
82. Safety Argument for MPS Feedback Level.....	156
83. Safety Argument for MPS Feedback Level – Example Case Study.....	157
84. Safety Argument for FMS Feedback Level.....	158
85. Safety Argument for FMS Feedback Level – Example Case Study.....	159

List of Figures (Concluded)

Figure	Page
86. Safety Argument for GLAW Feedback Level	160
87. Safety Argument for GLAW Feedback Level – Example Case Study.....	161
88. Safety Argument for CLAW Feedback Level	162

List of Tables

Table	Page
1. Project PoP	6
2. ARP4754A Failure Severity Classifications.....	13
3. Design Assurance Level and Failure Conditions (from DO-178C).....	14
4. Longitudinal Case Study: CType I, II and III Values	71
5. Defined Distances for Collision Avoidance Function	124
6. Proposed Required Mode Property Specifications	147

Acknowledgements

This 3-year project was foremost a team effort. The accomplishments made in this program were a result of close, often weekly interactions between Barron Associates, our AFRL customers, and our academic consultants. Matthew Clark was the AFRL program manager for this effort, and he brought in a number of respected researchers from other AFRL branches, NASA, and corporate contractors that all offered significant contributions to the ideas, methods, and approaches developed in this work. His efforts are greatly appreciated, and we would like to extend our gratitude to the members of this team who each offered many good ideas and unique insights into the problems addressed. Runtime assurance (RTA) is a complex scientific undertaking, but one that can potentially offer great rewards for future implementations of highly advanced systems.

We would also like to acknowledge the contributions of Uma Ferrell of Ferrell & Associates Consulting, Inc. She is a designated engineering representative (DER) for the Federal Aviation Administration (FAA) and provided consultation regarding federal regulations and aircraft system certification processes. RTA will require special considerations and approvals from governing regulatory bodies, and her insights in how to get started proved invaluable.

1 Executive Summary

This report presents the accomplishments made in a 3-year Air Force-funded SBIR Phase III program that focused on furthering the technical maturation of the concept of RTA.

Advanced, future systems of Air Force interest will include features such as 1) adaptation; 2) reconfiguration; 3) intelligent autonomy and real-time machine learning; 4) mixed/variable initiative man-unmanned machine interaction; 5) distributed, cooperative command and control architectures; and 5) hybrid cyber-physical systems. These capabilities will require ever increasing complexities in the inherent algorithm designs that are coded and housed in aerospace systems, both in the onboard processors on aircraft as well as the ground support infrastructures that will enable operational deployment of these advanced systems. For such systems to be deemed safe to operate, their designs will be required to go through rigorous analysis processes to be officially certified for deployment (either through civil and/or military governing authority). However, current accepted verification and validation (V&V) practices and analysis methods cannot address the complexities and unique features of these new, advanced designs. Adaptation relies on current environmental conditions, and many learning approaches are nondeterministic in nature. Such systems are impossible to fully analyze at design time, and it is impossible to explore, study or simulate every possible state or outcome such systems will exhibit when exposed to the infinite possibilities of real-world scenarios, unforeseen events and unanticipated environmental conditions.

The past decade has seen significant advances in design-time V&V techniques, such as formal methods, that can provide important benefits to the process of assuring these advanced systems operate safely and correctly during operation. However, it is widely accepted that runtime monitoring of these advanced systems will also be necessary to account for unforeseen states that simply cannot be predicted ahead of time and may pose potential safety hazards if encountered during deployed operation.

RTA is the process of monitoring key critical parameters during operation of an advanced system that cannot be fully certified at design time due to its inherent complexities. Using certain strategic tests, the RTA system determines whether unsafe conditions are ensuing due to some error in the advanced system. If this is the case, then the RTA monitor deactivates the advanced system and switches operation to a reversionary, trusted system to mitigate any subsequent unsafe condition. Since the reversionary system is trusted and certified at design time, it will not have the greater capabilities that the advanced system possesses, but should have the minimum required capabilities to recovery safe operation and, for example, return the platform to a safe location for recovery and post-mission analysis. In this manner, RTA protection *bounds* the behavior of the untrusted, advanced system, allowing it to operate and provide the benefits of its advanced capabilities, but disallowing any unforeseen, unsafe actions that could compromise system safety.

Barron Associates has been involved in the development of RTA systems for aerospace applications for over a decade, mainly funded through Air Force Research Laboratory (AFRL) programs, but also through a number of NASA-funded programs as well. Barron Associates' past efforts focused on development of particular, isolated RTA designs for an inner-loop control

system for an unmanned aircraft system (UAS) and for an outer-loop autoland guidance system for an optionally piloted UAS vehicle.

Because of the current key interest in future systems with higher level intelligent autonomy, a main objective of this Phase III program was to investigate RTA framework considerations for these higher level feedback loops. We developed a challenge problem to address this program objective that involved a multi-vehicle fleet of UASs performing some type of complex mission, working together in a distributed, cooperative command and control protocol. Although these are uninhabited systems, critical safety considerations can be of paramount importance. A failure or error in onboard software could lead to civilian casualties on the ground. Or, failure to accomplish the mission could lead to endangerment of ground troops. UAS platforms may also play a critical role alongside piloted aircraft, requiring highest levels of safety assurance.

The command/control architecture for each vehicle in the fleet is made up of four main feedback loops. As with all aerospace platforms, each vehicle has an inner-loop control system and an outer-loop guidance system. The guidance system is then driven by a higher level feedback loop termed the flight management system (FMS). The function of the FMS is to solve for a safe commanded path for the vehicle that is deconflicted from its neighboring fleetmates and other hazardous objects. The FMS is then commanded by a further higher-level feedback loop termed the mission planning system (MPS), which provides continually updated mission plan solutions, including future objective locations, required arrival times, asset allocation plans, etc. No single vehicle is designated as the fleet leader. All the vehicles act together in an interactive, cooperative manner. The MPS solution is a result of intra-fleet communication and negotiation amongst all members of the fleet. Each vehicle offers its own solution based on its current sensor input, current knowledge, and current situational awareness. Nominally, at each update to the mission plan, all vehicles share their disparate current information and iterate until all agree on the next update to the mission plan. The final fleet mission solution is derived from a fusion of this negotiation process.

Advanced, adaptive, intelligent autonomous functionality was considered to be present at any or all of these four nested feedback loops in this hierarchical, interacting framework. Hence, at each feedback level, RTA protection is required if that level employs an advanced, untrusted element. One of the key contributions of this program centered on how these four nested feedback loops interact and influence each other and the implications of these interactions on their RTA system designs. This program developed a unique, modular framework in which each feedback level could either be an industry standard design time assured (DTA) system, or could be a RTA protected system that could be in either advanced or reversionary operational mode. The requirements for the input/output architecture for such a system were explored. Also investigated were the implications of switching from advanced to reversionary mode at one level and the effects that can have on other levels. It was determined that the RTA monitoring process should involve more than just checking for system safety at the current feedback level, as degraded system performance at one level can have safety implications at other levels due to the inherent interdependence of each feedback level. Because of this, it was determined that an overall RTA monitor manager needs to continually pass operating mode information to each of the feedback levels so that each level is always aware of whether other feedback levels are operating in advanced or reversionary mode. Reversionary systems may have reduced

performance capabilities, and this information needs to be communicated with the other feedback levels to ensure overall correct and safe operation.

Another main contribution from this effort involved extensions to key ideas first developed by researchers at Carnegie Mellon University in the mid 1990s, who established what is known as the simplex framework. Although the reversionary system will not possess all the performance capabilities of the advanced system, it will be required to take control at any state in the operating region of the advanced system. The simplex framework involved a special controller that could take over at the switching condition and drive the system state to an operating region of a baseline controller. The baseline controller is defined to have established performance capabilities, but may have a comparatively restricted operating envelope. By using this special controller, the RTA system, along with its reversionary mitigation, would not severely restrict the operation of the advanced controller as it would have the capability of safely taking over operation at any state in the advanced system's operating region. We further developed this idea by allowing the reversionary system to be composed of any number of these special controllers, we denoted as transition controllers or systems, and any number of baseline controllers or systems, whatever is required to fully cover the operating region of the advanced system. We explored design methods for inner-loop transition controllers that could guarantee progress of the state from the switching condition boundary to the operating region of a chosen baseline controller. Numerical simulations studies provided clear demonstration of the benefits of RTA systems, showcasing the operation of these transition and baseline controllers in mitigating unsafe conditions caused by errors in an advanced control system that would otherwise cause catastrophic loss of the vehicle.

Another key contribution was the refinement of what we termed type safety definitions. These formal definitions of safety levels addressed some of the key research questions that were left unresolved from our previous RTA programs. Here, we defined three safety levels. Type I safety involves the actual or physical safety of the system under control or management. Type II safety involves the ability of the reversionary system to be able to take over operation and recover acceptable safety margins in a safe and timely manner. Type III safety addresses the required update rate of the RTA monitoring function to ensure that ensuing unsafe conditions are always identified before there is any possibility of entering into an unsafe state. If the RTA monitor detects that the Type III safety is lost, then this defines the switching condition – the RTA system must command a switch to the trusted reversionary system. By these definitions, safety is then always assured. We developed the Type I – III safety definitions in general as well as in particular for each feedback level.

The actual construction of the Type III switching condition boundary can be a complex, difficult and labor intensive task, depending on the complexity and number of critical states of the system under development. This could involve extensive analysis and simulation studies, requiring substantial and costly engineering labor hours. We did not fully address this problem in this program, but did perform some initial investigations into formal methods and software tools that could predict the state reachability without requiring extensive simulation experiments. In this manner, if the physical safety boundary can be obtained from heritage engineering design and analysis, then the switching condition boundary may potentially be constructed using these state reachability tools. This would give the required margins needed to ensure operational safety is

guaranteed through the control mode switching process from the advanced to the reversionary system.

Any future fielding of the nested, hierarchical feedback structure of the challenge problem we addressed will require design time certification. Because of the complexity of this challenge problem, we investigated an analysis approach known as compositional reasoning, which involves the construction of assume-guarantee (A-G) contracts for sub-elements or subsystems within the overall feedback system of the aircraft platform. These A-G contracts essentially define the constraints and design requirements for each element in each feedback loop structure. We started by constructing A-G contracts at the inner-loop controller level. Once these were completed, the entire inner-closed-loop level could be *rolled up* into one single composite block with its own composite A-G contract. This was then used in forming the A-G contracts at the guidance law feedback level. Once these were completed, this loop as well was rolled up into a single composite block that included the composite block of the inner-loop level. This procedure was then repeated for the FMS and MPS levels, resulting in A-G contracts for each level in the overall interacting system of feedback levels. Another key contribution of this project was the recognition that RTA systems must act as A-G contract monitors, ensuring that these contracts are always upheld. If there is a breach of contract for whatever reason, then this breach is mitigated by employing the appropriate reversionary controller or reversionary actions. The A-G contract checks that must be performed by the RTA monitor involve more than just safety checks at the current feedback level. They must also include required performance checks, checks on input and output validity, and hardware status checks including sensor and information integrity checks. These additional checks are required due to the complex inter-connectivity of the integrated, nested feedback architecture we addressed in this project's challenge problem.

One of the most important considerations for future autonomous aerospace systems is their safe integration into airspaces that include manned or piloted aircraft, as well as operations over congested, populated regions. Trusted automated ground and airborne collision avoidance systems will be essential to enable this integration. Another contribution of this effort was to investigate how to best design a framework that integrates interacting certified collision avoidance systems with the nested, hierarchical RTA protected framework. One approach is to have a completely separate and dedicated collision avoidance feedback framework that takes control of the vehicle at all feedback levels when a collision threat is detected. Once the collision avoidance procedure is completed and the threat has passed, control is then handed back to the RTA protected systems. Another framework is to have the certified collision avoidance systems as an integrated part of the RTA monitoring and reversionary systems, where threat determination may be a part of the FMS RTA monitor, avoidance maneuver generation a part of the FMS reversionary system, and avoidance guidance and control a part of the reversionary guidance and control law systems. In either framework, we note that the collision avoidance function should be fully trusted and certified to the highest criticality level. It is recommended that advanced, untrusted collision avoidance systems not be entertained, even with RTA monitoring.

Another set of contributions accomplished in this effort addressed the design time certification of the RTA protected framework. Prototype fault tree analysis (FTA) and failure modes effects and criticality analysis (FMECA) models were developed that directly addressed the fault mitigation functions of the RTA systems. Also, results of the compositional reasoning and A-G contract

construction were used to form the basis for a preliminary safety case argument using goal structuring notation (GSN) diagrams. An argument pattern was developed that could be applied at each feedback level. It is hoped that constructing safety case arguments will aid in developing specific artifacts and evidence that can be used in the required certification processes for eventual fielding of RTA protection on complex systems. Related to this effort, at the end of the program Barron Associates met with FAA DERs and sought their feedback on the topics of RTA protection and the safety case argument approach. Their initial response to our requests was positive in that they saw potential benefits for both RTA systems and safety case arguments. However, they also provided valuable feedback on technical and certification protocol hurdles that will need to be addressed in follow-on efforts.

Design, development and implementation of RTA systems are not solved problems. A number of key areas require further maturation. First, developing feasible switching condition boundary construction techniques will be required. Although we made great progress in formally defining the switching condition objectives, we have yet to solve the curse of dimensionality associated with the highly complex, multi-dimensional nature of the hypersurface boundaries that define the reversionary switching decision. Currently, this requires labor intensive, costly simulation exercises and extensive analyses. However, there seem to be promising mathematical approaches and software tools for reducing this burden and it is recommended that these approaches be further pursued.

Related to the curse of dimensionality problem is the development of methods for streamlining construction of reversionary systems. If the design and development of the reversionary systems are required to be so complicated that they negate the advantages of using the advanced system, then this will severely limit the benefits of the RTA protection and risk mitigation approaches addressed in this program. This problem was directly addressed at the inner-loop level, but it is recommended that further attention be paid to this area of research and development.

The nested RTA protected feedback loops have a high degree of interaction and the implications of switching to reversionary modes at one or more levels should be explored further. If some of the feedback levels are allowed to continue to run in advanced mode, while other levels are running in reversionary mode, then this framework of *graceful degradation* can have the potential for many unforeseen emergent behaviors, especially for heterogeneous fleets operating in any number of mixed-mode combinations. It is recommended to explore this topic further and to construct guidelines for how such disparate platforms can seamlessly work together in either advanced or reversionary modes at different feedback levels.

The complexity of the systems that we explored for RTA protection resulted in complex RTA designs themselves. If such systems are to eventually be certified for operational deployment, then the topic of certifying complex RTA protected systems must continue to be addressed. It is recommended that more interactions be pursued with FAA DERs and counterpart DoD certification authorities to alert these governing bodies to the ideas of RTA protection and to seek their feedback and guidance on how to construct RTA systems that can be design-time certified.

2 Introduction

2.1 Overview of Program and Report

This is the final technical report of the program, Contract No. FA8650-12-C-3227, entitled “Run Time Assurance Framework Development for Highly Adaptive Flight Control Systems.” This was a 38-month AFRL-funded SBIR Phase III program; however, a 3-month no-cost extension was granted in June, 2015, extending the period of performance (PoP) to 41 months. The following summarizes the milestone dates for the program:

Table 1. Project PoP

Milestone Date	Original PoP	PoP with 3-Month No Cost Extension
Start Date	26-Sep-2012	26-Sep-2012
End of Technical Effort	30-Aug-2015	30-Nov-2015
DFTR Due Date	30-Sep-2015	30-Dec-2015
End Date/FTR Due Date	29-Nov-2015	29-Feb-2016

This program consisted of a Base effort and an Option I effort and the results from both these parts of the program are presented in this final report.

This effort built on the research accomplished in the Phase I, II and Phase II Enhancement programs (see contract FA8650-05-C-3504), as well as two associated programs referred to as the Challenge Problem Initiative (CPI) and Challenge Problem Demonstration (CPD) programs, in which Barron Associates served as a subcontractor to Lockheed Martin (see PO Contract: 7166989 (CPI) and 7184702 (CPD)). More recently, an academic level research program was funded by AFRL to address some of the technical issues regarding RTA implementations, learned through the aforementioned efforts [Clark 2013]. This effort also built on findings of that program.

Barron Associates, Inc. was the prime contractor for this program with John Schierman acting as the principal investigator (PI) from Barron Associates. Matthew Clark was the AFRL program manager (PM) for this effort. Scott Smolka and Scott Stoller from the State University of New York at Stony Brook and Andre Platzer from Carnegie Mellon University acted as consultants on this contract.

The style and formatting of this document adheres to the required style designated by the Air Force’s Scientific and Technical Information Office (STINFO).

2.2 Motivation and Background

Mission and safety requirements for current and next-generation aerospace vehicles have given rise to flight software with an ever-increasing level of complexity and autonomy. *Intelligence* is built into new and emerging designs through the development of novel algorithms that detect, learn, adapt, switch modes, coordinate, plan, etc. For manned vehicles, the complexity of safety monitoring, weapons management, mission planning, envelope cueing, flight control, and other systems is such that sophisticated software is required to assist the human pilot in accomplishing the mission. Fully autonomous and uninhabited systems or remotely piloted systems require an

even greater level of sophistication and autonomy since the human pilot (capable of intelligent contingency planning and reasoning) is no longer a part of the system in situ. Further, human-machine cyber physical systems must have the intelligence to adjust their level of autonomy in real time, depending on changes in the system, environment, or other conditions. Under such scenarios, the control software should complete selected tasks automatically, allowing the human operator to take on more of a managerial role.

The current process of verifying the system software involves exhaustive offline testing of every possible state of the software code. Although this has worked well for classical control systems and heritage software, current-generation controllers are already reaching a level of complexity that pushes the envelopes of existing V&V approaches, with little hope for affordable V&V of next-generation intelligent systems. As the complexity of flight controllers grows linearly, the cost associated with V&V has grown exponentially and it is widely recognized that applying today's V&V practices will be an untenable undertaking for adaptive, learning and nondeterministic algorithms, which are common characteristics of future guidance, control and trajectory/mission planning strategies (Buffington 2003). Such algorithms will be extremely difficult and costly, if not impossible to fully certify at design time with current V&V methods.

The increasing level of complexity in advanced, adaptive and autonomous systems has generated the requirement for advances in the technologies used to certify these new systems. In 2004, AFRL embarked on the Flight Critical System Software Initiative, funding the Certification Techniques for Advanced Flight Critical Systems (CerTA FCS) and other related programs [Storm 2008], [Hollingsworth 2010]. These programs were intended to advance the state of the art in V&V techniques through formal methods approaches for design-time or offline analysis, and runtime monitoring (or RTA) for operational protection of systems running untrusted code that is too complex to be fully certified at design time. A general description of RTA is given in the next subsection.

It is expected that through the combined use of new advances in design-time V&V approaches along with the use of RTA systems during online operation, the system behavior can be provably bounded [Schierman 2014(a)], [Schierman 2008], [Rudd 2009], [Aiello 2010]. There are a number of advanced systems and systems-of-systems concepts that are of great interest to the Air Force (as well as NASA and the other branches of the Department of Defense (DoD)) and it is for these reasons that there is interest in further developing the RTA concept so these advanced systems can be certified for fielded operations. In June of 2015, the DoD's Office of the Secretary of Defense published a memorandum on its technology investment strategy for the V&V of autonomous systems [DoD 2015]. In that memorandum it listed several main goals for investment, one of which was runtime behavior prediction and recovery. The DoD recognizes that even with advancements in design time analysis methods, the most complex autonomous systems will require runtime monitoring along with mitigation strategies for undesired decisions and behaviors of the automated intelligence elements in these advanced systems.

Barron Associates has been involved in RTA development efforts through the SBIR program. Initial studies focused on developing runtime monitoring systems for aircraft inner-loop control systems and the approach evolved into a framework that is strikingly similar to the simplex architecture, developed by researchers in the computer science field at Carnegie Mellon University [Seto 1998], [Bak 2011]. Barron Associates was subsequently involved in a

Lockheed Martin effort in which a runtime monitoring approach for an outer-loop guidance autoland system was developed [Aiello 2010], [Storm 2008]. This project entailed a substantial effort to develop a ground-laying *implementable* runtime safety assurance approach. However, at the end of this project, several technical hurdles were identified, and these are discussed in Section 2.7. These technical challenges were a main motivating factor to further develop RTA technologies in the current Phase III program.

2.3 Definition of RTA and the RTA Protected System

There are many different approaches to runtime monitoring or RTA frameworks. A general framework is presented in this section, with more details on how RTA can be applied to aerospace platforms given in the next chapter. Further details are presented in subsequent chapters that focus on particular feedback loops. Consider the following definitions for design time assurance (DTA) and RTA:

DTA: the process of performing V&V analysis and testing of software *offline*, at design time (that is, before *live* operation) to the level required for certification of the plant or system the software is housed on. The required certification level depends on the defined criticality of the software, which is determined by the level of hazardous effects that errors in the software can have on the plant. Required certification levels are defined in more detail in Subsection 2.4.1. If a set of software can be fully V&V'd to its defined required certification level at design time, then that software is considered *trusted* for live operation on the plant. That is, there is an *acceptable level of confidence* that the software will operate correctly to the required certification level of the system within which it operates. If a set of software cannot be fully V&V'd to its defined required certification level (due to its complexity, nondeterminism, etc.), then that software is considered *untrusted* (or does not have an *acceptable level of confidence*) for live operation on the plant. Untrusted software may also arise during developmental testing stages due to its experimental nature, in which full V&V analysis/testing would be cost prohibitive at that point in the design cycle.

RTA: the process of monitoring a system containing untrusted software during runtime or live operation of the plant to determine if the untrusted software is operating correctly. If it is determined that the untrusted software is not operating correctly or anomalous or unsafe behavior is detected, then to mitigate any adverse effects that may ensue, operation of the untrusted software is terminated and control is switched to trusted reversionary counterpart software. That is, the RTA system activates some type of recovery action to ensure continued safe or correct operations. The presumption here is that the reversionary software has equivalent basic functionality as the untrusted software, but would not have all of its advanced capabilities. The reversionary software would be able to continue safe operation of the plant or system, but at a reduced level of performance, capability, or functionality.

One early application of RTA was to protect local computer networks from errors in server software upgrades [Seto 1998]. The RTA system would rapidly revert to the prior, proven server software if any erroneous operations were observed. This was not, however, a safety critical application of RTA. Again, here we are concerned with applying RTA to safety critical systems in aerospace applications. In this case, safety is ensured by initiating a recovery process that employs a trusted reversionary system (one that is design-time V&V'd to the appropriate certification criticality level). The RTA monitoring function observes the state of the system (or

certain critical parameters) while the advanced software is activated and determines if either 1) the current state lies within a predefined safe operating region or envelope, or 2) the state of the system will leave the safe operating region if control is not switched to the reversionary system. If the first condition is true, then the advanced, untrusted software is allowed to continue to operate. If the second condition is true, then control is switched to the reversionary system.

2.3.1 RTA Protected System Framework

The concept of an *RTA protected* or a *runtime protected system* (or subsystem) is introduced here. This is shown in Figure 1. A runtime protected system is considered to be the fundamental element in an overall RTA design. This is presented here in a universal or generalized manner, applicable to any feedback level (i.e., inner-loop control, outer-loop guidance, etc.) or any system in general. Therefore, in the figure, it is shown that input to the RTA protected block can come from upstream sources (or commands into the block) and from feedback from one or more downstream sources.

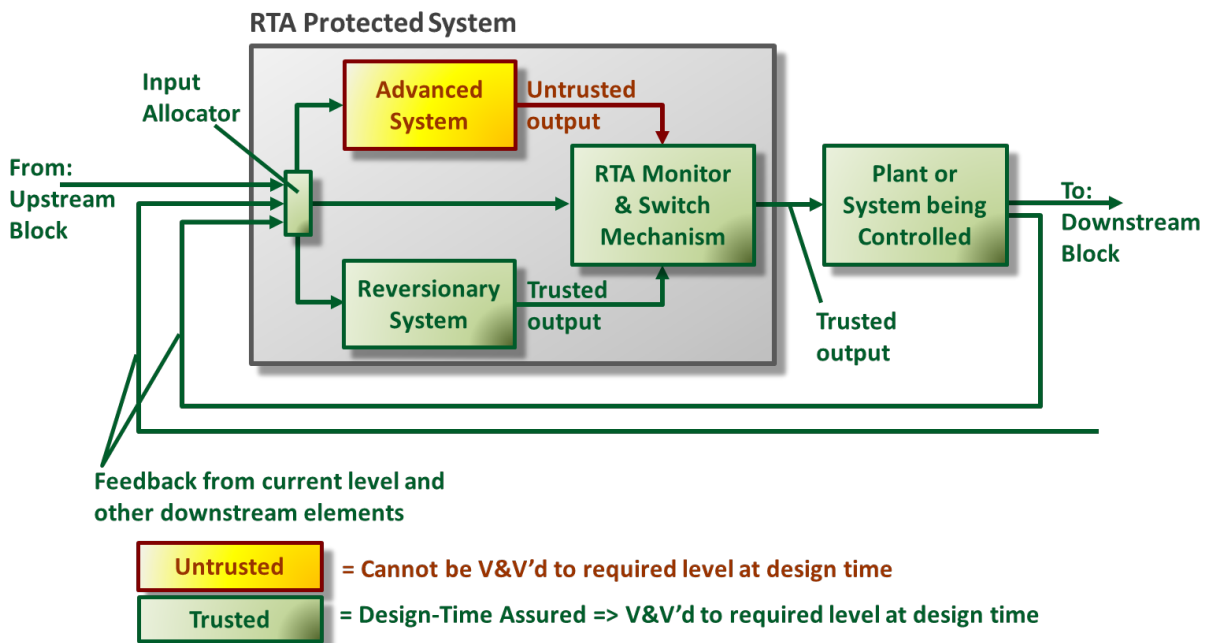


Figure 1. The Concept of an RTA Protected System

It is noted in the figure that the advanced system and its output are *untrusted*. The advanced system cannot be fully V&V'd to its required certification level at design time. All other blocks in the figure are *design-time assured*. That is, they are fully V&V'd to their defined required certification level at design time. This includes the plant being controlled or operated, the input allocator, the reversionary system, and the RTA monitor and switch mechanism block. Hence, all other information flow is considered *trusted*. All input information into the RTA protected block is trusted, and, most importantly, all information out of the RTA protected block is trusted. Even if the output of the advanced system is passed through to the plant (because no adverse conditions were detected at the current time), the claim here is that that output is trusted because it has been checked by the trusted RTA monitor. Ultimately, the goal is to make a safety case

argument that the RTA protected system is equivalent in terms of safety to a system that is fully V&V'd at design time with accepted analysis/testing practices. That is, even though there is an untrusted element within the RTA protected block, all output downstream from that block will be fully trusted information. We note here that throughout the report we often use the term *certified* when describing an element, block, subsystem or set of code. What is meant by this is that element, block, etc. is fully V&V'd at design time to the required level for certification of the system within which it is housed. Or, that that element is trusted to the required level. We recognized that the legal definition of the term *certified* (used by the governing regulatory commissions), is different than how we use it in this report. By the legal definition, *only systems are certified*. Hardware and software are not certified; rather, they are approved *within the context of a system that is certified*.

To give visual context to the overall framework developed in this program, the following figure shows the upstream/downstream connections for the RTA protected systems for the multiple nested feedback architecture of one UAS platform. Again, the challenge problem addressed in this program was multiple UAS platforms performing a mission under a cooperative framework, where each feedback level could potentially be a RTA protected system. This figure shows, for example, that the upstream information going into the RTA protected guidance system (or GLAW block) comes from the FMS block, and the information generated by the RTA protected GLAW block is delivered downstream to the RTA protected control system (or CLAW block). More detailed discussion of this system is presented later in the report.

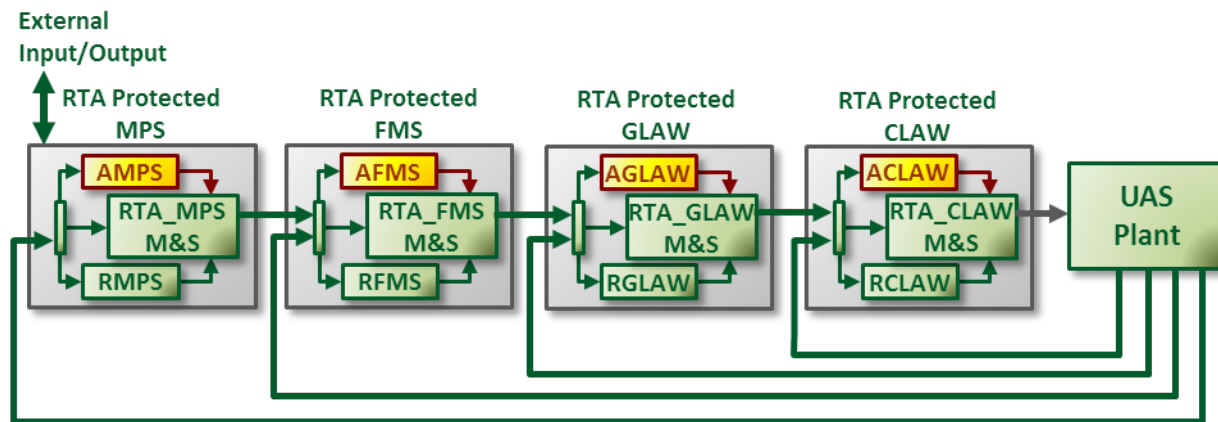


Figure 2. RTA Protected Systems in the Overall Framework

2.4 Current V&V and Certification Processes

For readers who are unfamiliar with this topic, the V&V processes for certification and the main purpose and functionality of RTA systems in assuring safety are presented here.

2.4.1 Review of Industry Standards and Regulations

Standard and regulations adopted by the FAA for civilian commercial and general aviation operations will be considered here. Although missions and requirements of DoD operations will obviously have differences with civilian air travel, safety requirements will be similar and comparable safety practices have been adopted across all branches of the DoD (see, for example, MIL-HDBK-516B, Feb. 2008, Department of Defense Handbook, Airworthiness Certification

Criteria, which encompasses use of MIL-STD-882, system safety standard practices). Furthermore, for DoD systems, evidence and arguments generated using civilian FAA airworthiness standards are considered acceptable for DoD standards

Code of Federal Regulations (CFR) have various titles and sections, and Title 14 sections govern aeronautics and space. These rules have been formulated to maintain safety of flight. Rules cover all aspects of aviation such as airports, lighting, security, cargo restrictions, aircraft design, aircraft manufacturing, aircraft parts, aircraft maintenance, training of all of the personnel (pilots, controllers, mechanics, etc.). Rules are updated as technology demands such updates or when an incident or an accident occurs suggesting a possible improvement. Different definitions for *parts* define the Rules for Airworthiness and Safety for different types of aircraft (e.g., Part 23 includes commuter, utility, etc. aircraft; Part 25 includes transport aircraft, Part 27, 29 involve rotorcraft, Part 33 is for aircraft engines, Part 35 is for propellers, etc.). For each of these parts, these rules cover their function, equipment, systems and installation. In general, these CFRs impose that every equipment, systems and installation:

- Performs intended function
- Is absent from any unintended function
- With no single point of failure
- Has consideration of differing levels of potential hazards onboard the aircraft

SAE Aerospace International is an organization that provides industry standards and guideline documents that have been adopted by the FAA via advisory circulars for the aerospace industry. These documents are developed, drafted and revised by a number of committees that are made up of industry, regulatory and academic subject matter experts and specialists in the particular areas addressed by the committees. One of the main guiding documents used by the aerospace industry is ARP4754A [ARP4754A 2010], which provides recommended practices for aircraft and system development processes. Associate Advisory Circular (AC 20-174, Development of Civil Aircraft and systems, 2011) recognizes ARP4754A as an acceptable method for establishing a development assurance process. ARP4754A describes the development process taking into account the operating environment and the functionality thus including validation of requirements and verification of design implementation for certification and process assurance. This development assurance process and the guiding doctrine for each part of this process are shown in Figure 3. This figure shows the general process of using the various industry documents to incorporate known safety information specific to the intended operation in the development of *validated requirements* which are then used to develop and verify using specific hardware and software processes using the “DO” documents. DO documents are developed by RTCA special committees of stakeholders from industry, regulatory bodies and academia and codified by using the Advisory Circulars. RTCA, Inc. a not-for-profit organization that produces technical guidance documents and acts as an advisory body to the FAA.

Software housed in onboard processors for airborne vehicles is, like all other parts of an aircraft, considered a subsystem within the overall platform. All subsystems on aircraft, both hardware and software, require full *approval* to the appropriate criticality level (defined later) at design time (before operational deployment of the aircraft) in order that the aircraft system be certified. A full certification of flight or safety critical software code requires both verification and also validation of the code, defined as:

-
- ```
graph TD; IF[Intended A/C Function] --> SAG[Safety Assessment Process Guidelines & Methods (ARP 4761)]; SAG --> ASDP[Aircraft & System Development Processes (ARP 4754A / ED-79A)]; ASDP --> SAG; ASDP --> F[Function, Failure & Safety Information]; F --> SAG; ASDP --> SDI[System Design Information]; SDI --> SAG; ASDP --> FS[Functional System]; FS --> O[Operations]; O --> SASCS[Safety Assessment of Aircraft in Commercial Service (ARP 5150 / 5151)]; IMAG[Guidelines for Integrated Modular Avionics (DO-297 / ED-124)] <--> EHLCE[Electronic Hardware Development Life Cycle (DO-254 / ED-80)]; IMAG <--> SDLC[Software Development Life Cycle (DO-178C / ED-80C)]; EHLCE <--> ASDP; SDLC <--> ASDP;
```

ARP 4754A, along with [ARP 4761 1996] are used to address the validation of the algorithm design and [DO-178C 2011], “Software Considerations in Airborne Systems and Equipment Certification” is used to address the verification of the code implementation of that algorithm design. Traditionally, verification of the software code and validation of the design are accomplished through exhaustive testing and simulation, and through extensive fault analysis methods. Full approval to the appropriate criticality level requires both verification of the onboard software code and validation of the algorithm design.

Approved for public release; distribution unlimited.

potential to cause only up to a hazardous failure condition, then that component need only be assured to Level B, and so on.

**Table 2. ARP4754A Failure Severity Classifications**

| <b>Severity Classification</b>        | <b>Minor</b>                                                                                                                                                             | <b>Major/Hazardous</b>                                                                                                                                                          | <b>Catastrophic</b>                                          |
|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------|
| Failure condition effect              | <ul style="list-style-type: none"> <li>• Slight reduction in safety margins</li> <li>• Slight increase in workload</li> <li>• Some inconvenience to occupants</li> </ul> | <ul style="list-style-type: none"> <li>• Significant reduction in safety margins</li> <li>• Significant increase in workload</li> <li>• Some discomfort to occupants</li> </ul> | Failure conditions prevent continued safe flight and landing |
| Development Assurance Level           | Level D                                                                                                                                                                  | Levels C and B                                                                                                                                                                  | Level A                                                      |
| Probability objective per flight hour | 1.0 to 1.0E-5                                                                                                                                                            | 1.0E-5 to 1.0E-7                                                                                                                                                                | 1.0E-7 to 1.0E-9                                             |

The criticality levels for the verification process of software code come from DO-178C and are analogous to the development assurance levels listed in Table 2. These are presented in Table 3, where the *software level or the design assurance level* (DAL) is defined by the severity of the failure. Analogous to the validation question, here too, if coding errors in the software can potentially lead to a catastrophic failure of the system, for example, then that code must be tested to the extent of being approved or trusted to DAL A. If errors lead only up to a hazardous condition, then the software needs only be assured up to DAL B, and so on.

ARP 4754A dictates the development assurance level each software component should be assigned based on intended functionality. ARP4754A presents guidelines on developing a functional hazard assessment (FHA), which identifies failures and error conditions categorized by severity at both the aircraft and system (or subsystem) level. The FHA also generates an environmental and emergency configuration list and derives safety requirements for the design at each level. The FHA leads to the generation of the preliminary system safety assessment (PSSA), which provides a complete failure conditions list and generates safety requirements in compliance with FHA requirements. These include operational and lower level safety requirements. The final step is then the system safety assessment (SSA), which provides a comprehensive analysis of all safety implementations. This includes an updated failure condition list with rationale showing compliance with safety requirements (both qualitative and quantitative). SSA documentation includes proof of how requirements (safety, protection, etc.) for the design of system installations have been incorporated; evidence used to validate failure condition classifications; required maintenance tasks to maintain safety; etc. Another important part of the safety assessment process is the common cause analysis (CCA), which verifies that all failures identified by the SSA are independent from each other in the system under evaluation. This ensures that there are no single points of failure in redundant or recovery architectures.

Much of the tools used in developing the SSA come from ARP 4761, which provides guidelines and methods on the safety assessment process for aerospace systems. Safety assessment

processes using tools and techniques noted in ARP 4761 lead to the identification of what can go wrong and what can contribute to an unsafe condition.

**Table 3. Design Assurance Level and Failure Conditions (from DO-178C)**

| <b>Design Assurance Level (DAL)</b> | <b>Failure Condition</b> | <b>Description</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------------------------------|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>A</b>                            | Catastrophic             | Failure Conditions which would result in multiple fatalities, usually with the loss of the airplane.                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>B</b>                            | Hazardous                | <p>Failure Conditions which would reduce the capability of the airplane or the ability of the flight crew to cope with adverse operating conditions to the extent that there would be:</p> <ul style="list-style-type: none"> <li>- A large reduction in safety margins or functional capabilities</li> <li>- Physical distress or excessive workload such that the flight crew cannot be relied upon to perform their tasks accurately or completely, or</li> <li>- Serious or fatal injury to a relatively small number of the occupants other than the flight crew.</li> </ul> |
| <b>C</b>                            | Major                    | Failure Conditions which would reduce the capability of the airplane or the ability of the crew to cope with adverse operating conditions to the extent that there would be, for example, a significant reduction in safety margins or functional capabilities, a significant increase in crew workload or in conditions impairing crew efficiency, or discomfort to the flight crew, or physical distress to passengers of cabin crew, possibly including injuries.                                                                                                              |
| <b>D</b>                            | Minor                    | Failure Conditions which would not significantly reduce airplane safety, and which involve crew actions that are well within their capabilities. Minor Failure Conditions may include, for example, a slight reduction in safety margins or functional capabilities, a slight increase in crew workload, such as routine flight plan changes, or some physical discomfort to passengers or cabin crew.                                                                                                                                                                            |
| <b>E</b>                            | No Effect                | Failure Conditions that would have no effect on safety; for example, Failure Conditions that would not affect the operational capability of the airplane or increase crew workload.                                                                                                                                                                                                                                                                                                                                                                                               |

From ARP4761, an aerospace platform is viewed as a system of systems or subsystems, each of which has a stated or defined function. This document defines:

- Function - intended behavior of a system based on a defined set of requirements regardless of implementation.
- Failure - loss of function or a malfunction of a system or a part thereof.
- Error - 1) an occurrence arising as a result of an incorrect action or decision by personnel operating or maintaining a system, 2) a mistake in specification, design, or implementation.
- Hazards - potentially unsafe condition resulting from failures, malfunctions, external events, errors, or a combination thereof.

Each step of the development process/product is fed directly or indirectly from these safety decisions and mitigations. The assumptions and identification of unsafe conditions is a known knowledge base that grows with every incident and accident with new items added as designs, techniques and materials change. Tools and techniques defined in ARP 4761 include:

- Fault Tree Analysis
- Dependence Diagram
- Markov Analysis
- Failure Modes and Effects Analysis
- Failure Modes and Effects Summary
- Common Cause Analysis
- Zonal Safety Analysis
- Particular Risks Analysis
- Common Mode Analysis

Two important analyses from ARP 4761, FTA and FMECA, are explored in more detail specifically for systems with RTA in Appendix E.

Requirements captured using ARP 4754A thus result in a set of *validated* requirements per known safety factors for that specific design with considerations of operational and environmental factors, design related assumptions, manufacturing methods, serviceability and installation. It should also be noted that the requirements from this point onwards are treated as a closed loop system which is monitored for violations throughout the design and development of the system implementation (using software or hardware) and integration.

The current certification model is that the validation of system properties are accomplished at the system level via system safety assessments using known safety challenges and assumptions. Thus the requirements that are allocated to components of a system are validated requirements. Requirements are then treated as a *closed system* which demands that any new items or deviations are justified – these new items and deviations are captured as *derived requirements* which in addition to being justified are scrutinized at the system level for safety.

Software which has been allocated a portion of the validated requirements from system processes must accomplish these functions with demonstrable precautions to show that:

- a. Errors are minimized in the translation of requirements to design to code. Rigor in development and verification gives a level of comfort according to the DAL assigned by the system processes.
- b. No requirements have been left out and no *features* have been added. Assure that:
  - as-specified = as-built,
  - as built = as-verified, and
  - as verified = as-delivered.

DO-178C with its *leveling* has a formulaic process definition for development and verification for each DAL with the highest number of objectives imposed at the highest DAL. These are listed in Annex tables, which cover areas such as the software planning and development processes, verification of outputs of design and requirements, verification of coding and integration, integration testing, configuration management, quality assurance, etc. Each of these tables list a number of objectives and which ones need to be met by which DAL category. For example, for the software development processes, some of the objectives listed cover development of high and low level requirements and their provision to the system safety assessment process, development of software architecture, development of source code, etc. In



summary, for software at DAL A, 71 objectives must be met, 33 of which must be achieved by documented independent review teams (i.e., the person performing the analysis/testing cannot be the same person writing the software code, for example). For DAL B, 69 objectives must be met, 21 of which achieved by independent review, etc., with each successively less critical DAL having fewer required objectives to be met.

The basic idea at all DALs is to have a correct translation of validated requirements to implementation, and prove that there are no *extra features* that will cause safety problems. Compliance to DO-178C just demonstrates that the objectives defined for the process at its appropriate DAL have been accomplished and no problems were found by internal Quality Assurance or by an independent regulator. No direct safety statements can be made about software that is compliant to DO-178C. Safety is assured through system certification, which are accomplished processes conducted at the system level, as dictated by ARP 4754A.

## **2.4.2 Runtime Assurance for System Safety**

RTA monitoring is checking for anomalous conditions that can lead to unsafe states. These conditions could be a result of either an algorithm design flaw or a software coding error. Let us define the term *criticality level* as encompassing the assurance requirements of both validation (the development assurance level definitions from Table 2) and verification (the analogous DAL definitions from Table 3).

For example, an inner-loop controller will command control surface actuation, which, in turn, controls the attitude of the aircraft. Failure of that inner-loop controller can cause unstable attitude, loss of control and potential catastrophic loss of the aircraft, resulting in potential loss of life as well. Therefore, inner-loop control software will be designated at *criticality level A*.

In general, inner-loop control systems will be safety critical and will be assigned *criticality level A*. Outer-loop guidance systems may be assigned *criticality level B*, flight management and mission planning systems may be assigned *criticality level C*, depending on the particular platform design. However, since we will be addressing unmanned systems and fleets of interacting vehicles, presumably performing critical military missions, the required safety assurance levels for each feedback element may all be designated *criticality level A* or *B*. For example, failures in the guidance or flight management levels could lead to civilian casualties on the ground. Or, failure to accomplish the mission could lead to endangerment of ground troops, depending on the criticality of the mission objectives.

For safety critical systems on aerospace platforms, the number of analysis, testing and process objectives that must be met is quite rigorous, as it should be. However, new advanced control, guidance and planning systems characterized as having adaptive or intelligent autonomy with complex and/or nondeterministic algorithms may not be able to meet all required objectives at criticality level A or B, for example, given current V&V practices. Adaptive systems use critical feedback gains that are a function of the current operating environment, which results in an infinite number of possibilities for feedback control characteristics. New planning algorithms that explore search spaces starting from random initial conditions or genetic algorithms that explore candidate solutions through random mutations are gaining wide interest for their powerful numerical capabilities. However, by their very nature, it is impossible to fully predict every possible outcome that results from the range of expected inputs encountered during typical

missions, let alone adverse, unexpected or extreme conditions. For advanced and complex systems, several formal methods tools exist that can automatically perform V&V code checking to a certain extent. The verification process of uncovering coding errors has met with greater success than the validation of the design using this class of tools. Validation of the algorithm design remains the more difficult task, as it is difficult to assure that all potential fault pathways have been determined and mitigated [Manson 2003]. Nevertheless, for complex systems with advanced designs, coding errors and fundamental design flaws may not be discovered during design time V&V procedures because they may have subtle or little effect under typical system operation. However, their effects can be substantial under rare or unforeseen conditions.

For this reason, runtime monitoring has been deemed necessary to add that additional layer of protection for such hard to certify advanced systems, and the research community has been investigating RTA as a means to provably bound the characteristics of the algorithm/software that cannot be certified to its required criticality level. Safety monitoring considerations are covered in DO-178C, reproduced below:

### **“Safety Monitoring**

Safety monitoring is a means of protecting against specific failure conditions by directly monitoring a function for failures that would result in a failure condition....

Through the use of monitoring techniques, the software level of the monitored software may be assigned a software level associated with the loss of its related system function. To allow this assignment, there are three important attributes of the monitor that should be determined:

- a. Software level: Safety monitoring software is assigned the software level associated with the most severe failure condition category for the monitored function.
- b. System fault coverage: Assessment of the system fault coverage of a monitor ensures that the monitor’s design and implementation are such that the faults which it is intended to detect will be detected under all necessary conditions.
- c. Independence of the function and monitor: The monitor and protective mechanism are not rendered inoperative by the same failure that causes the failure condition.”

Therefore, for example, DO-178C allows for advanced code that has a failure criticality of DAL A to operate even if it can only be verified or trusted to DAL B<sup>1</sup>. To allow this, the monitoring function (in our case, RTA) must be a) trusted to DAL A, b) it is proven that all failure conditions resulting from all faults of the advanced software are correctly detected under all operating conditions, and c) all such failure conditions cannot cause failure of the monitoring function.

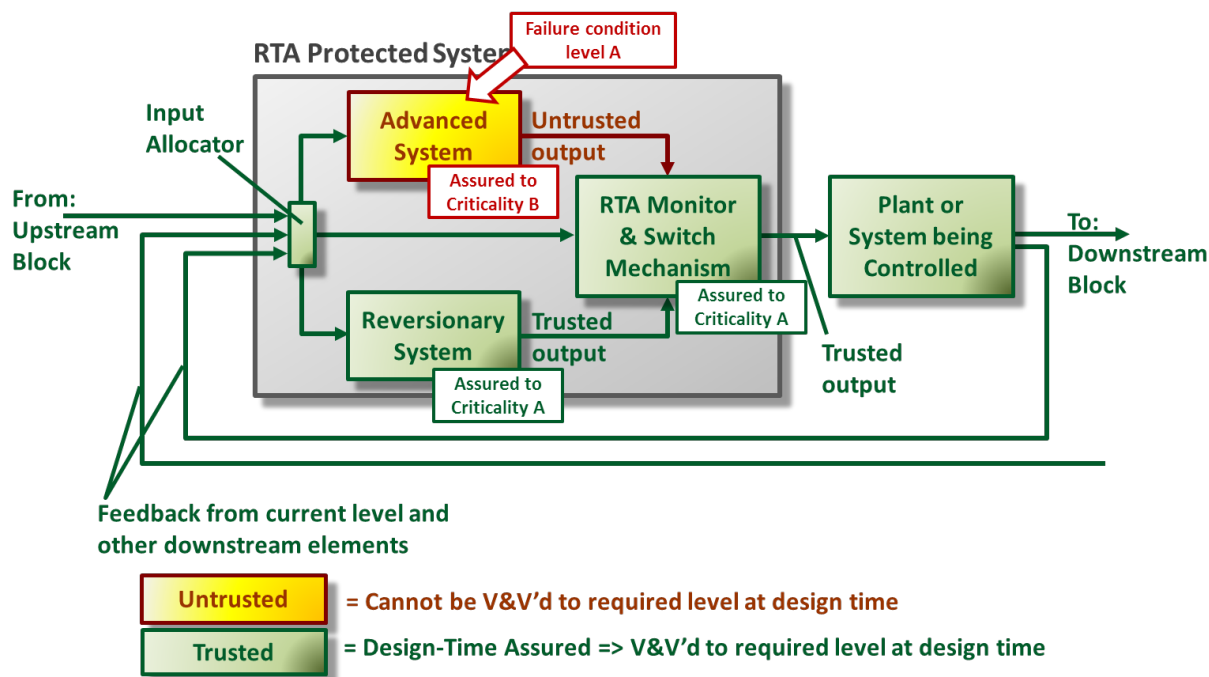
---

<sup>1</sup> Current regulations from ARP 4754A state that the less trusted software can only be one DAL less than the monitoring software. This may cause difficulty in certifying systems with RTA protection as the advanced, untrusted system may not be able to be assured as high as one DAL below the subsystem within which it resides. Special considerations and further trust in RTA systems will hopefully ease this regulation in the future.

Attribute a) is illustrated in Figure 4, which is a repeat of Figure 1, but now with example criticality assignments for each block. Again, the advanced system can only be V&V'd to criticality level B (or lower if allowed), but faults in the advanced system can potentially lead to a failure condition of level A. For this reason, the RTA monitor and switch block and the reversionary system must both be V&V'd to criticality level A (the input allocator and any other code that reside within the RTA protected system must also be V&V'd to criticality level A as well).

Attribute b) is intricately related to the assurance to criticality level A for the RTA monitor and is really the validation requirement for the RTA monitor. That is, the detection of faults and the decision condition to switch to the reversionary system must be fully validated (to level A) to operate correctly at all times. This topic is covered in detail in the next chapter.

For attribute c), one example would be that if faulty inputs into the advanced system cause a failure condition, and those same inputs are sent to the RTA monitor, then some type of detection mechanism must be in place that determines the inputs are faulted and the RTA monitor should not rely on such information. This too will be discussed in the next chapter, in which we discuss that all information into the RTA monitor must be trusted information if we are to trust the RTA protected system as a whole. This may require some type of fully certified hardware or sensor health monitoring system, for example.



**Figure 4. The RTA Protected System and Criticality Level Defined Software**

### 2.4.3 RTA Safety Case Argument

Again, RTA (as developed in this program) does not directly perform verification or validation. Rather, it covers both verification and validation indirectly by checking for adverse conditions resulting from *either or both* software coding errors or algorithm design flaws. We make the

argument here that key to certifying an overall system that includes an uncertifiable subsystem (i.e., the advanced system that cannot be fully assured to the required criticality level) is that inclusion of a properly designed and implemented RTA system will correctly determine unsafe conditions at all times and correctly mitigate the event with a trusted subsystem (i.e., the reversionary system) in a safe manner at all times. Therefore, employing RTA should provide the safety guarantee artifacts (safety constraints and properties) that can be used toward a successful argument that it accomplishes an equivalent level of safety to that of a traditional, design-time assured system.

RTA and the advanced, adaptive software it is monitoring cannot be approved using traditional certification guidance. However, we propose that the problem be viewed from a system safety perspective and an argument of *equivalent level of safety* be made using strong engineering practices and a plausible safety argument. Such an argument may be made using the foundational code of federal regulations:

- 14 CFR §xx.1301 Function and installation
- 14 CFR §xx.1309 Equipment, systems, and installations.

In general the safety case for RTA must show that the system as a whole - not just the RTA system – 1) performs its intended function, and 2) is absent from any unintended function.

That part of a complete *safety case argument* that focuses on how RTA systems provide continued safety through monitoring and recovery actions will be presented in Chapter 10. Safety case arguments are a new *tool* in the certification process, yet to be widely adopted by the industry or regulatory governing bodies. The safety case argument approach is gaining wide interest because it is designed to complement current safety standards through the construction of a structured and defensible argument that provides the evidence and artifacts necessary for certification.

## **2.5 Program Objectives, Scope, and Challenge Problem**

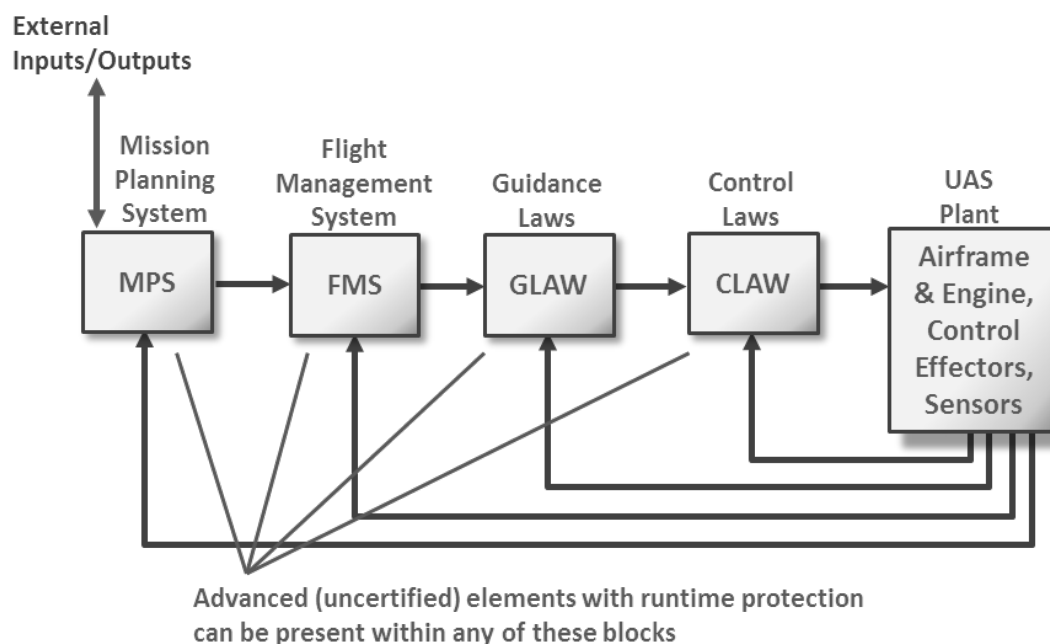
Our focus is on highly complex, advanced systems characterized as having capabilities such as adaptation, learning, or intelligent autonomy. Hence, by their very nature, such systems would require large engineering organizations with many designers and developers to construct an actual full-up application integrated with RTA systems. Given the size of the current program, such an objective was clearly out of scope. Therefore, it was not the intent to construct a complete, detailed RTA design. Rather, the main objective of this effort was to cultivate suggested best practices and approaches for designing and developing RTA systems and how they would interact and operate in advanced applications. We therefore explored candidate RTA frameworks and candidate design approaches for constructing and integrating RTA systems within the overall plant. The results of this project lay the groundwork for follow-on real-world design efforts that will involve detailed considerations in the design and development of RTA systems. This project provides guidance on including RTA design considerations early in the development of the overall feedback architectures for such future aerospace systems.

In our original SBIR Phase II effort, the focus was on developing an isolated RTA system at the inner-loop control level, and in the aforementioned CPD program we focused on developing an isolated RTA system applied to an outer-loop guidance system in autoland mode. However, at

the outset of this program, the Air Force had strong interest in future systems with intelligent autonomy and other complex, advanced capabilities. Since we had already investigated application of RTA to inner-loop control and outer-loop guidance systems, there was more interest to explore application of RTA to intelligent planning algorithms and mission and flight management systems, which would reside *upstream* at a higher feedback level than the control and guidance feedback levels. Therefore, we developed a feedback framework and challenge problem demonstration system that would address these outer feedback loops and considered this for an unmanned aircraft system (UAS) platform.

Figure 5 shows the general architecture for this multiple, nested hierarchical feedback system. Advanced systems with runtime protection will be considered present at any or all of the feedback levels shown in the figure. Note that this is a very simplified view of the overall feedback system and we will add more detail of the function and role of each block in later chapters.

The inner-loop controller is for a UAS with morphing wing capabilities. The function of the inner-loop control law (CLAW) is to maintain attitude stability and to accurately follow guidance law (GLAW) commands. The function of the GLAW is to generate commands for the inner-loop such that the vehicle follows a path defined by a series of waypoints. The function of the flight management system (FMS) is to generate the waypoint locations and airspeed commands so that the vehicle flies a planned trajectory to a certain location by a certain time, defined by the mission plan, which is generated (or updated during runtime) by the mission planning system (MPS).



**Figure 5. Nested Feedback Architecture for Challenge Problem**

This feedback loop hierarchy works in an integrated fashion and the design of RTA systems at some or all feedback levels will need to address how the other feedback levels will influence and

affect the operation of the RTA protected levels. Again, since we did not investigate application of RTA systems at the FMS and MPS levels in past programs, the original plan was to place greater emphasis on these levels in this program and the majority of the base effort was to focus on the FMS and MPS levels and have the Option I effort focus on maturing the RTA framework at the GLAW and CLAW levels. However, we discovered early in our investigations that all feedback levels needed to be investigated because of the inherent interactions due to the feedback pathways. The Base effort mainly funded the RTA framework development and design considerations for the overall multi-nested feedback system and how the RTA elements can be used to develop a full system safety case as a means toward certification. The Option I effort funded the generation of experimental results using a specific inner-loop controller design for a specific UAS platform that had morphing wing capabilities. This model was chosen to investigate particular aspects of the reversionary controller that were of interest to the team and addressed a technical hurdle identified at the end of the Phase II program.

It is interesting to note the relationship between RTA mitigation or reversion decisions and the failure condition categories listed in Table 2. Because of the complexities of the interactions between the different feedback levels, reversion decisions do not necessarily have a direct correspondence to the failure condition categories in this table. For example, if the inner-loop control system is rated at criticality level A, the inner-loop RTA monitor does not simply switch to the reversionary controller only when catastrophic failure conditions are ensuing. Rather, it may switch due to lesser failure conditions for other reasons. As an example, a failure in the inner-loop control system that leads to loss of attitude stability would be considered a catastrophic failure condition, as it would result in the rapid loss of the vehicle itself. The RTA system in this case would have to immediately activate recovery actions. However, a failure in the inner-loop control system that leads to poor trajectory path tracking may be considered only a loss of performance at the inner-loop level, but would be considered a loss of safety at the guidance level as it could potentially result in a collision with a neighboring vehicle or a ground obstacle. Therefore, the failure condition is catastrophic at the guidance level and in the other upstream feedback loops as well. Further, even minor failure effects that indicate software errors imply that the software system being monitored by the RTA can no longer be trusted, and recovery actions should be enacted as those errors could potentially cause catastrophic effects later in the mission.

For this project, we will generally define loss of safety as any error condition that can be detected by an RTA system that has the potential to lead to loss of continued safe flight or loss of continued safe mission operations. This may encompass minor to catastrophic failure conditions at the current feedback level, but because the RTA monitors are not acting in isolation, the failure condition effects at other levels must be taken into account as well.

The challenge problem investigated in this program also involved fleets of UASs performing a cooperative mission. An instantiation of the MPS is installed on each vehicle in the fleet, which performs fully autonomous decision making through intra-fleet communication and negotiation with the other fleetmate vehicles. We assume here a cooperative, decentralized and distributed command/control/communication architecture. Each vehicle has the freedom to plan its own path, but in a coordinated manner with its neighboring fleetmates. This coordination is performed at the FMS level, again through intra-fleet communication. The function of the MPS

is to perform the mission planning during runtime, negotiating with fleetmates for task assignments, objective locations, arrival times, contingency plans, etc.

Note that this interactive architecture is in contrast to a fleet flying in a tightly coordinated *formation*, with a single vehicle acting as the formation *leader*, and all other vehicles simply acting as *followers*, performing station-keeping guidance in which they each maintain a certain slot position with respect to the leader's position. Such formations can often be viewed as a single entity since only the leader needs to plan out where to go to meet the objectives of the mission. We did not consider UAS fleets flying in such *leader-follower formations* in this project as this would be considered a lesser certification challenge than the distributed mission planning architecture that we addressed.

In summary, one of the main contributions of this effort is that we have expanded the design focus of runtime protected systems from isolated studies of a single feedback level to multiple, nested feedback levels, each with their own RTA system. We further matured the RTA technologies by broadening their application to a wider class of challenge problems, involving multi-vehicle mission planning and morphing wing UAS platforms.

## **2.6 Technical Approach**

### **2.6.1 Key Assumption on New Approach**

The frameworks developed in this effort represent new architectures for feedback systems that directly incorporate RTA systems. Our key assumption here is that advanced inner-loop controllers or advanced outer-loop guidance systems or mission planning/flight management systems may have to be restructured (possibly radically restructured) to successfully work in conjunction with an RTA system and such considerations will need to be addressed early in the design cycle.

*This is an important point to make. Barron Associates began this project with the mindset that we were attempting to design a retrofit RTA system. That is, the designers of the advanced code dictate the structure of the advanced code (be it inner-loop control, outer-loop guidance, etc.) and the RTA system then needs to be constructed to protect the system from errors in the advanced software. However, this new viewpoint opens up much more design freedom in constructing RTA protocols. That is, now RTA designers have more freedom to explore how the advanced code should be constructed or organized to work better with the RTA system. One key benefit here is that if, early in the design stages of the system, RTA protection is considered integrated with advanced components, then this can help to generate the required assurance arguments needed in the final certification process.*

### **2.6.2 Generalizing the RTA Architecture**

In this project, we have attempted to generalize the key elements of the RTA architecture so that its *blueprint* can be applied or customized to all feedback loops under study (inner/outer/flight management/mission planning). Clearly, there will be certain loop-specific aspects and even design-specific features that will be required. While we are not claiming to have developed a *one size fits all* approach, the general RTA framework presented in this report should be applicable to most classes of safety critical applications, certainly in aerospace platforms, but

also in other application areas such as industrial processes, medical devices, nuclear power plants, etc.

## **2.7 Review of Technical Challenges in RTA System Development**

A number of key areas for further research and development of RTA technologies were recommended at the conclusions of the more recent AFRL funded projects (the aforementioned CPI and CPD programs) and these are discussed below. Brief descriptions of how these challenges have been addressed in this project are also presented. However, not all of these topics have been fully addressed in this project and this list provides suggested areas for further research and development.

### **2.7.1 Performance Limitations**

The RTA approach developed in our prior efforts limits the performance of the advanced system to that of the reversionary system's safe-to-fly envelope. This was not considered a limitation in our past projects because the main focus was on adaptive control systems, in which the advanced features enabled stability and control of the system under control effector failures or other system damage. Therefore, the purpose of the advanced elements was not to expand the operating envelope. However, this limitation needed to be addressed for general advanced systems which potentially have expanded regions of operation. Therefore the simplex architecture approach [Seto 1998], [Bak 2011] was reviewed, which addresses this limitation through the utilization of a *safety controller* with a larger operating envelope. Its purpose is solely to take the state from the ensuing unsafe region to the operational envelope of the pre-certified baseline system. This approach is discussed in detail in Appendix A.

We have further expanded on this approach by considering that the reversionary system may be made up of a number of transition controllers (akin to the safety controller concept in the simplex framework), the purpose of each is to successively drive the system state to the next transition controller until a safe state is reached in which one of potentially several baseline controllers can then take over. Notionally, the baseline controller gives the system a *return-to-base* capability, whereas the transition controllers recover from the approaching unsafe condition. The reversionary system is composed of the complete set of transition controllers and baseline controllers and, if properly designed, should cover all or almost the entire operating region of the advanced system. In this manner, the reversionary system should be able to take control if warranted anywhere in the operating envelope of the advanced system. This is not to say that the reversionary system has the same level of performance capabilities or other functional capabilities that the advanced system exhibits (otherwise, since the reversionary system can be fully certified at design time, then there would be no need to employ the advanced system). The reduction in performance or other capabilities that results from switching to the reversionary system needs to be taken into account during operation and this topic will be discussed in further detail in later chapters of this report.

### **2.7.2 Safety Boundary Construction**

In our past projects, we formed safety constraints in terms of boundaries in state space. The boundary between safe and unsafe state space regions is typically defined by physical or functional (i.e., control limitations) design aspects of the system in question. For aircraft, this boundary is defined, in part, by maximum wing loading (a structural constraint), maximum



angle-of-attack before stall (an aerodynamic constraint), control moment limits, control rate and deflection limits, engine operating limits, etc. All of these limits are functions of environmental conditions, Mach number, altitude, flight condition, etc. They may be further influenced by the current vehicle configuration, such as whether the landing gear are deployed, the amount of remaining fuel (which affects mass moments of inertia), or whether the vehicle is carrying stores or payload. Finally, note that the safety boundary is further defined by control system design constraints of the reversionary system's certified operating region, which can potentially limit operation from reaching actual, physical safety constraints (to allow for safety margins, this will typically be the case). In summary, the safety boundary can be a complex, multi-dimensional hypersurface, potentially discontinuous for hybrid systems with both continuous and discrete state aspects. Hence, these boundaries are not easily described, visualized or constructed for complex systems.

In our past CPD project, the approach to constructing the safety boundary was simulation intensive to the point of being potentially infeasible. The process was a labor and computationally arduous task and the "state space explosion" problem limited how much we were able to define. We relied on extensive offline simulation analysis to construct the safe operational region, which was then interrogated online in a fast, computationally efficient query algorithm. The main contribution of the effort was the online components; however, offline, we quickly encountered the "curse of dimensionality" problem in constructing the safe envelope database. Even a small number of state variables can cause the simulation requirements to grow exponentially and we had to limit our studies to a subset of the true number of states critical for safety assurance. Had this been a larger industry effort, constructing an actual runtime monitoring system, then it is possible that enough funding and engineering staff hours could accomplish the job. Yet, the problem in the CPD project only addressed an autoland system – the last few moments in the flight of the aircraft. How to feasibly expand the simulation approach to the full operating envelope of the system remains an open research question.

One approach is to investigate whether some closed-form functional relationships exist between certain states or critical parameters that define safety relationships. If so, this set of such functions could be continuously evaluated online to determine the current safety condition, potentially reducing the amount of required offline simulation evaluation for the remaining states/critical parameters that determine overall safety. However, this approach certainly requires domain expertise and will be highly application specific.

In a related SBIR Phase I program that recently completed, Contract No. FA8650-14-M-2456, entitled "Combined Approaches for Verification and Validation of Run Time Protected Systems," [Schierman 2014], we began to address a formal process for construction of the safety envelope using a targeted, multi-step simulation approach. Here, a coarse grid is first used to construct an estimate of the safety region. Subsequent simulations then focus on more precisely determining the safety region's boundary with finer gridding of the simulation's initial conditions around the coarse estimate of the boundary. This may be an iterative process until the required precision is achieved. The boundary is then "filled in" using certain methods borrowed from machine learning techniques, such as support vector machines and signed distance function methods. Final verification of the boundary is then accomplished using forward and back reach methods.

In [Clark 2013], Pappas and Kumar investigated the construction of safety guarantees also looking at state reachability using zonotope methods for linear systems. They also considered Hamilton-Jacobi methods and construction of barrier certificates for nonlinear control systems. Barrier certificates define regions in the state space which the system cannot reach and safety is assured if all unsafe states lie only in such regions. They also investigated the reduction of simulation requirements through the construction of bisimulation functions and surveyed a number of software tools that aid in targeted simulation searches.

Other construction methods with more formal, theoretical underpinnings should be investigated as well, as these may hold the promise of determining safety boundaries without the need for labor intensive simulation studies. One example is presented in [Speltzer 2015] which exploits continuation methods using numerical bifurcation analysis. Further maturation of this and other approaches are left for follow-on efforts.

### 2.7.3 Switching Condition Determination

The safety boundary and the switching condition (the point at which control should be switched to the reversionary system) are not necessarily the same. Switching to the reversionary system at the safety boundary could potentially cause the system state to enter unsafe regions of the state space. This is because 1) control mode switching can often cause transient responses, 2) physical systems have momentum, so state trajectories cannot be instantaneously turned back toward safe regions (i.e. there will be overshoot into unsafe regions), and 3) modeling uncertainties are always present, so errors or inaccuracies can cause the state to actually be unsafe.

At each online query of the RTA system, the RTA monitor must ask: “if the advanced system is allowed to continue to operate for  $T$  seconds into the future, will the reversionary system be able to safely and successfully take over control at time  $T$  if necessary? If the answer to that question is “no” then the RTA system must switch to the reversionary system at the current time. The general research question is how much margin should there be between the switching condition and the safety boundary to ensure the system always remains safe (e.g., what should be the value of  $T$ ?). This margin may be determined offline, again through simulation or other formal analyses, or partly online through runtime prediction methods.

We addressed this problem in this project by constructing formal definitions of three levels of safety boundaries. The details of these definitions will be presented in the next chapter, with particular variants for each feedback level in following chapters. However, in summary, the first level of safety addresses the actual safety of the system. This could deal with physical safety (stability, structural integrity, etc.) or functional safety (functions working correctly, etc.), depending on the system being controlled. The second level addresses the ability of the reversionary system to be able to successfully recover the system without unsafe events ever occurring. The third level of safety addresses the update rate requirements of the RTA monitor to ensure that the advanced system cannot cause an unrecoverable condition between RTA monitoring updates. These safety level definitions formally determine the switching condition that ensures the required margin or look-ahead time is properly defined. If the state of the system violates the third level of safety, then the RTA system activates the reversionary system, and by definition, the reversionary system has the capability and enough time to recover safe operations.

**Online Prediction Options:** In this and past projects, we have assumed the switching condition boundary is constructed offline in some manner (simulations, formal methods, etc. – see Subsection 2.7.2). Online prediction methods may also be employed to provide *look-ahead* information that may help to reduce the required margins between the first level safety boundary and the switching condition because the predictions will use current measured states and operating condition information [Bak 2014]. That is, more accurate, up-to-date information may give more accurate estimates of ensuing unsafe conditions. Further, online prediction may potentially lessen the labor intensive offline simulations required by reducing the needed accuracy in the switching condition. Online high fidelity nonlinear predictions may be too computationally burdensome for typical flight processors and may not be able to perform in real time. Simple linear prediction schemes may offer the best solution. We did not address online prediction in this project and further study and analysis is needed if online prediction schemes are pursued (see Chapter 13 for a list of recommended follow-on studies).

**Statistical Modeling:** There will always be uncertainty and error in system models used to define boundaries or to perform online predictions - even with high fidelity models. Therefore, some margin is needed in defining the envelope boundaries due solely to uncertainties. However, no formal research has been performed to accurately quantify such margins. Assumptions and simplifications that allow faster run times imply that more margin is required in defining the envelope bounds. Quantifying the margins will most likely be a function of these assumptions and uncertainty models. In our original Phase II SBIR program [Ward 2009] we developed a preliminary on-line linear prediction approach that introduced uncertainty bounds through probability models.

It is recommended that further maturation of statistical modeling be performed even for nonlinear system models. Barron Associates has completed a number of programs that employ the generalized polynomial chaos framework [DeVore 2010] to model probabilistic uncertainties in dynamical systems. This approach may be applicable here and should provide for better estimates on the uncertainty bounds for prediction methods and envelope margins.

**Control Mode Switching Process:** Further study is needed to explore technical issues regarding transferring control to the reversionary system. A formal analysis is needed to develop bounds on transients introduced in the transfer process. At the inner-loop level, generalized methods for integrator match up should be investigated that ensure integrator wind-up is avoided and control effector rate and deflection limits are not violated. The study of the bumpless transfer control problem may be applicable in this case [Arehart 1996], [Graebe 1996]. Outer-loop and flight management/mission planning mode switching may also demonstrate technical hurdles and should be investigated as well.

#### **2.7.4 Vehicle Health and Contingency Management**

Many advanced controllers are adaptive to be hardware fault tolerant. The net result of a hardware fault (control effector failure, vehicle damage, etc.) may be to drive the system state into an unsafe region. If the adaptive controller is designed properly, it should be able to successfully recover safety of the system from the faulted condition. Depending on the severity of the fault, after successful adaptation, the vehicle may recover full nominal performance, or may operate in a *crippled* state, requiring mission termination. However, the adaptive controller should have the ability to return the platform to a safe home airbase. Before completion of the

adaptation, or during the *crippled* operations, the vehicle may operate in what would otherwise be considered unsafe states.

If adaptive control systems are to be monitored by RTA systems, care must be taken that the RTA monitor does not improperly shut down the advanced controller because it observes unsafe states which are due to hardware failures or other physical damage, not any error in the adaptive controller. Therefore, if a hardware fault or physical impairment has occurred, this information needs to be relayed to the RTA monitor. The RTA system should then allow the advanced controller to continue to operate since it has the adaptive capabilities to recover from the hardware faults/physical damage, whereas the reversionary controller does not. At the other upstream loops, there may also be adaptive elements in the advanced versions that are designed to specifically work with the adaptive inner-loop controller. Here too, their respective RTA systems should then allow the advanced elements to operate if hardware faults have been recognized.

At the core of integrated vehicle health management (IVHM) systems are fault detection and isolation (FDI) algorithms. The function of an IVHM system is to determine if there is anomalous behavior due to damage to the physical system, control effector malfunction, or faulted sensors. Therefore, an IVHM system should be utilized if possible for runtime protected adaptive systems with hardware fault tolerant capabilities.

Likewise, RTA operations require accurate sensor information as it continually monitors the system for unsafe states. IVHM systems that can determine and isolate faulty sensors can then work in conjunction with a redundancy management system or some other component to either reroute the sensor signal pathways to working sensors, or analytically reconstruct the required information from other disparate sensors. If this cannot be accomplished, the IVHM system must – at the very least – inform the RTA system that certain information can no longer be trusted to be correct. Other recovery actions must then be taken.

If no damage or failures are detected by the IVHM system, then any anomalous behavior (unsafe states) observed by the RTA system will be due to software faults, and the advanced controller will be shut down and control will be switched to the reversionary control system. Because of the importance of the information provided by the IVHM system, it must be fully certified at design time to the required criticality level.

Although IVHM and FDI may be considered to be currently in research and development stages, substantial progress has been made in recent years with several significant development efforts by major airframe manufacturers, government entities and academic institutions (Boeing, NASA, AFRL, Cranfield University, etc.), [Srivastava 2009]. Such systems may not always be required for RTA applications; however, they may be commercially available for UAS platforms in the near future, and any benefits that can be provided by IVHM/FDI systems should be utilized by designers of RTA systems. Barron Associates has developed a unique approach to FDI [Ward 2009] and our technologies have matured over the past several years. Although not addressed in this project, integration of FDI methods with a developed RTA system should be studied further.

### **2.7.5 Design-Time V&V Methods for the RTA System**

Since our focus is on RTA systems monitoring safety/flight-critical software, we will need to investigate methods that can design-time certify the RTA software and the associated feedback system to the appropriate ARP4754A/DO-178C criticality level or an Air Force/DoD equivalent standard. The RTA system itself cannot be so complicated that it too cannot be certified at design time. Therefore, we seek approaches that can perform the key online checks for the highly complex system that assure operational safety, yet are simple enough to be trusted and certified before the system is fielded.

Current practices, such as the aforementioned FHA, PSSA/SSA, FTA, FMECA processes, etc. should be employed as a necessary first step toward certification of RTA systems. However, new approaches may also be required, such as combined simulation and formal analyses (Schierman1 2014), or state reachability tools studied in this program by our consultants, Scott Stoller and Scott Smolka of Stony Brook University, as well as in [Clark 2013]. We will also present a preliminary safety case argument later in the report, which uses a goal structuring notation (GSN) format to show that system safety is always maintained through the RTA monitoring and recovery process. This provides a further artifact that can be used for eventual certification of runtime protected systems.

### 3 Fundamentals of the Developed RTA Framework

This chapter will present the fundamentals of the RTA framework developed in this effort. We augmented and matured the basic ideas of the simplex framework to address the complexities in the challenge problem addressed. Again, the simplex approach is reviewed in Appendix A. From Subsection 2.7.1, the key to the simplex approach is that it includes an additional recovery controller that allows the feedback system to operate in the larger region of the advanced controller and we adopt that same strategy in this program.

#### 3.1 RTA Protected System

We generalize the block diagram presented in Figure 4 below, in which the RTA protected block must be assured to a criticality level X (X being A, B, or C), but the advanced system can only be assured to a criticality level Y, where Y is a lower level of assurance than X (Y being B, C, or D). Recall, we define the criticality level as encompassing both the development (validation) and the design (verification) assurance levels, as dictated by ARP 4754A and DO-178C, respectively. *The key here is that the RTA protected block increases the safety assurance to the required level at runtime.*

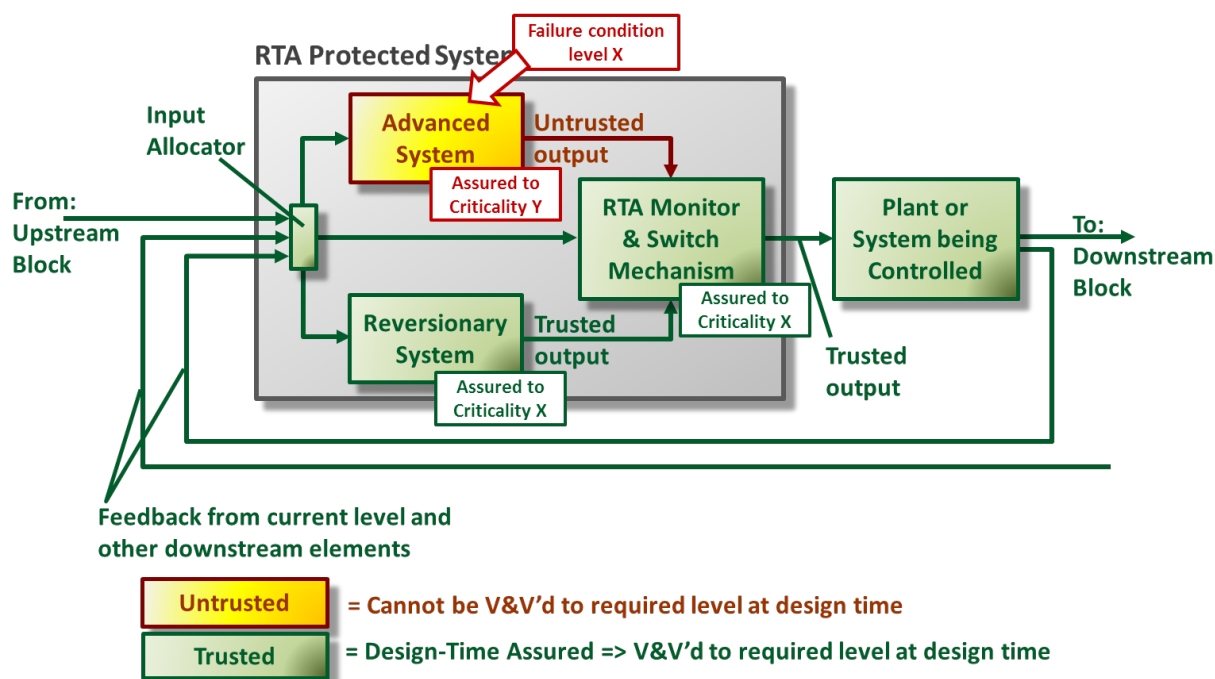


Figure 6. The Generalized Concept of an RTA Protected System

##### 3.1.1 General Description of Elements within the RTA Protected Framework

1. Plant or System being Controlled: Again, the plant or system being controlled in Figure 6 is a general representation of any system. It could be a physical system, such as the open-loop aircraft platform (including control effectors, airframe, engine and sensors), or a system with lower levels of feedback control, such as a closed-loop inner-loop control system, wrapped around the physical plant. In more general terms, the plant does not need to be a physical

system, and could be entirely cyber in nature. This could also include discrete or hybrid systems in which the plant could take on different modes of operation.

2. Inputs to the RTA Protected System and Input Allocator: The inputs into the runtime protected block include:
  - i. upstream commands,
  - ii. feedback from the current feedback level, and
  - iii. feedback from any downstream level, which may include other required information (such as system health monitor information or other information required by the RTA monitor).

The input allocator shown in Figure 6 parses these sets of inputs to their appropriate destinations. This will be discussed in more detail later.

3. Advanced System: This is the primary, full-envelope, full-featured controller or manager that is responsible for achieving advanced control or mission objectives, and may contain adaptive, learning or nondeterministic algorithms that cannot be certified at design time. In general, this element is considered so complex that it is infeasible or impossible to certify to the required criticality level with current V&V tools, such as exhaustive testing. Therefore, this system is not *fully* certified at design time (although it will most likely be V&V'd to a certain level as a natural part of the design and development process). Alternatively, it may be a temporary, experimental controller in which full a V&V process is cost prohibitive.
4. Reversionary System: This is the fail-safe system and represents a simplified counterpart system where the emphasis is on safety as opposed to performance. Again, analogous to the simplex architecture, we consider that the reversionary system is composed of a set of transition systems and baseline systems so that it can cover all (or almost all) of the operating region of the advanced system. These sub-elements of the reversionary system will be discussed in more detail later in this chapter.
5. RTA Monitor/Control Mode Switch Mechanism: This block takes in all necessary information available and performs certain key checks to determine if the system is approaching an unsafe operating condition. If so, it will switch control from the advanced system to the reversionary system, which will then recover the system to a safe state. What is checked and how it is determined that a control mode switch is necessary will be discussed next.

## 3.2 Formal Definitions of Safety Levels

One of the most important functions of the RTA monitor is to check for system safety. In this section we begin to refine the definitions of safety, first presented in Subsection 2.4.1. In our aforementioned SBIR Phase I program, we developed preliminary formal definitions of levels of safety at the inner-loop to address the question of what critical parameters should be monitored and when should control be switched to the reversionary system to always assure safe operations [Schierman 2014]. In this program, we have further refined these definitions and have applied them to each feedback level in the overall system. We present these definitions here in terms of a general system and will more precisely define them when we discuss each feedback level.

### 3.2.1 General Concept of Safety

The concept of being safe or operating in a safe manner or being in a state of safety may be well understood but difficult to actually define. Most definitions of safety rely on its antithesis – that of not being safe. In fact, the Merriam-Webster dictionary defines safety as 1) freedom from harm or danger, 2) the state of not being dangerous or harmful, or 3) a place that is free from harm or danger. Clearly, from ARP 4754A, this same approach is adopted since this document focuses on defining failure severity classifications (see Table 2), rather than on levels of safety assurances or guarantees. For our applications of interest in this program, here too we adopt this same approach in defining a safe state of the plant being controlled.

**Definition 1.** The set  $S$  - let  $S$  be the set of all states of the plant or system being controlled.

**Definition 2.** The set  $S_{safe}$  - let  $S_{safe} \subseteq S$  be the set of all states that are *safe* with respect to the plant being controlled. *Safe states* are operating points in which the plant functions as intended or designed. If the plant is a physical system, then states within the set  $S_{safe}$  will not cause or lead to:

- 1) environmental conditions that cause uncontrollable or upset conditions;
- 2) physical damage of the plant itself;
- 3) damage or adverse conditions to other plants or systems within the plant's influence; or,
- 4) harm or injury to human operators or other persons.

If the plant is a non-physical system (such as software code), or a more general hybrid or discrete system-of-systems under some type of mission management, then states within the set  $S_{safe}$  will not cause any adverse condition that prevents continued correct operation of the plant or any other interacting system or sub-system (in this case we make an equivalence between correctness of operations and safety).

### 3.2.2 Determined Safety (Dsafe Space)

It is often the case that the set  $S_{safe}$  cannot be precisely determined or defined. That is, its exact border with actual unsafe states is typically uncertain due to modeling errors, inaccuracies in measuring or estimating system states, changing environmental conditions, etc. For this reason, we typically add in *safety margins* to account for such uncertainties. Further, safety and the built-in safety margins are often a matter of judgment, defined by design teams or field operators of the plant in question and are often application, condition, and/or mission dependent.

**Definition 3.** The set  $S_{Dsafe}$  - let the set  $S_{Dsafe} \subseteq S_{safe}$  be the set of all states determined to be *safe* with respect to the plant being controlled. We assume the set  $S_{Dsafe}$  is defined by the team or organization that designed and constructed the plant and performed extensive engineering analyses (including simulation studies, hardware bench testing, flight testing, etc.) to determine where in the state space the plant can operate as intended in a safe manner. For physical plants, the border of  $S_{Dsafe}$  may be thought of as the set of *acceptable plant limits*. If the plant is an algorithm or process, for example, the border of  $S_{Dsafe}$  may be the defined allowable ranges of



critical parameters or variables used within the code that guarantee the operations will continue correctly without the process *crashing*.

Again, because safety margins are usually built into the design process, there may be states outside of  $S_{Dsafe}$  that are also safe, but are either not known to be safe, or considered too close to being unsafe for the plant to be allowed to operate at that state. So, although there may be safe regions outside of  $S_{Dsafe}$ , **the plant designers have defined  $S_{Dsafe}$  to be the *never-go-beyond* set within the state space  $S$ .**

As an example, consider an airframe that is analyzed through extensive wind tunnel testing. In this testing, its angle of attack (AoA) is incrementally positioned and the airflow around the structure is measured and analyzed. At say, an AoA of 15 degrees, it is determined that the airflow around the wings begins to separate. The airframe designers therefore deem that the vehicle should not fly beyond, say 14 degrees AoA for safety reasons. The vehicle may actually be quite capable of flying to, for example, 16 or 17 degrees AoA and still have enough attached airflow to maintain controlled flight. However, because of the uncertainty of unsteady airflow, the designers set a maximum AoA well below what will actually cause the wing to fully stall, which would lead to loss of controlled flight.

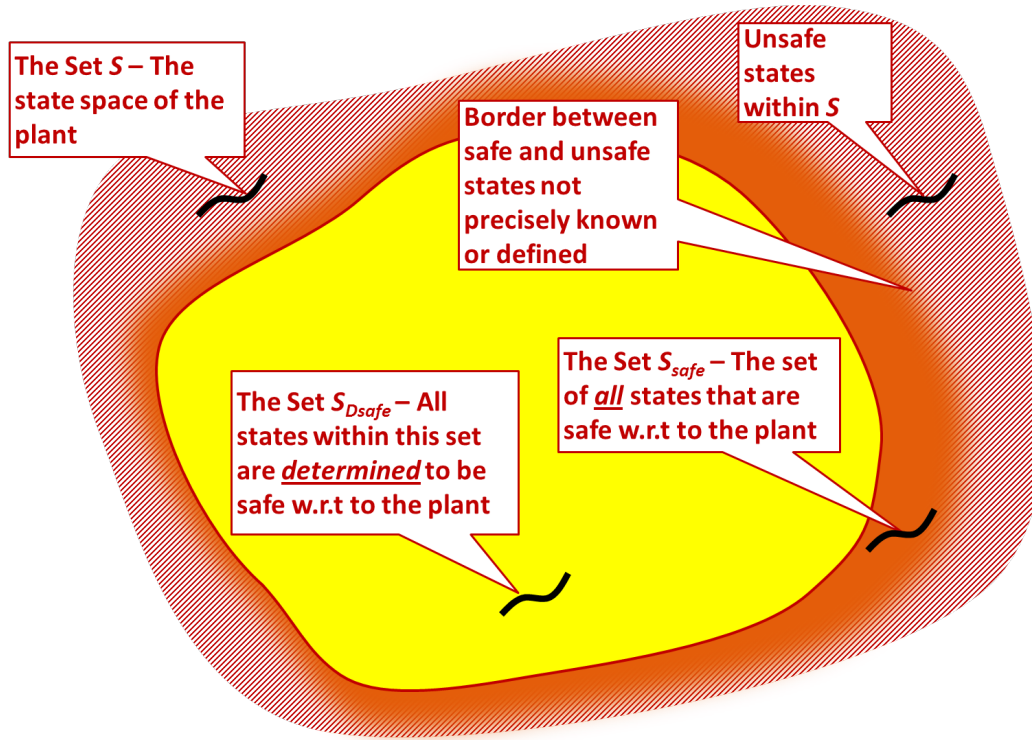
For aerospace flight control applications  $S_{Dsafe}$  is typically a complex, multi-dimensional hypersurface because its boundaries will be functions of several interdependent states/parameters (e.g., airspeed, altitude, AoA, sideslip angle, pitch/roll/yaw, air density, etc.). The set  $S_{Dsafe}$  can also be discontinuous and nonconvex, especially for hybrid systems-of-systems. Recall that we assume the plant or system being controlled is fully certified to the appropriate criticality at design time. Through the process of this certification, we assume here that the set  $S_{Dsafe}$  is defined/constructed and that its accuracy can be trusted to the same criticality level.

We note that the set  $S_{Dsafe}$  is a function only of the plant itself and is not a function of the controller or manager that is driving/directing the plant. Algorithm or coding errors in a poorly designed controller may have the potential to drive the plant operating from within  $S_{Dsafe}$  to a state outside of  $S_{Dsafe}$ , be it safe or unsafe. However, if the plant is in a state outside of  $S_{Dsafe}$ , no guarantees can be made that a controller exists that can recover the plant to a safe state inside of  $S_{Dsafe}$ .

Finally, note that  $S_{Dsafe}$  defines all states of the plant that can affect its safety. That is, it includes not just the typical dynamic states that may be used to build a simulation model, but also all plant configuration states and all environmental states as well. For example, the stall airspeed is a function of how much fuel is onboard, whether the vehicle is carrying a payload or stores, whether its landing gear is stowed or deployed, etc. So those *configuration states* are included in the definition of  $S_{Dsafe}$ . Likewise, the environmental state can affect safety and aircraft are rated for what type of environment it can safely operate within. For example, small UAS platforms may not be designed to fly in high wind or turbulence conditions. Such weather would therefore

be excluded from  $S_{Dsafe}$ . If a UAS does not have night vision sensors, then nighttime operations would not be allowed and only daytime or clear visibility conditions would be included in  $S_{Dsafe}$ .

The relationship between the sets  $S_{safe}$  and  $S_{Dsafe}$  is illustrated in the Venn diagram in Figure 7.



**Figure 7. Relationship between the Sets of Actual and Declared Safe States**

### 3.2.3 Safety Levels for Runtime Checking

With the set  $S_{Dsafe}$  defined, we can now formally define safety levels that are used to determine the switching condition protocol for an RTA protected system.

**Definition 4.** Type safety - A point  $x_0$  in the state space  $S$  is:

- **Type I Safe** if that point lies inside  $S_{Dsafe}$ .
- **Type II with set  $Q$  and time  $T$  Safe** if all of the following hold:
  - a) Point  $x_0$  is Type I safe,
  - b) Upon switching to the reversionary system, the state trajectory can converge to at least one point in a *desired* set  $Q$  within a given *desired* time  $T > 0$ ,
  - c) The state trajectory from the point of switching to the reversionary system to the point of reaching the set  $Q$  is entirely contained within the Type I safe region.

- **Type III with Period  $\tau$  Safe** if all of the following hold:
  - a) Point  $x_0$  is Type II safe,
  - b) Every possible output of the advanced system for a time period  $\tau$  results in a state trajectory entirely contained within the Type II safe region.

#### **Discussion of Definition 4**

Since Type I safety is defined as being the set  $S_{Dsafe}$ , it accounts for the safety properties of the plant itself. The RTA system should never allow the plant to operate outside of the Type I safe region.

Type II safety accounts for the behavior of the combined system consisting of the plant and the reversionary system. Its definition is such that, from any point in the Type II safe region, the RTA switch could activate the reversionary system, which can then maintain control over the plant, driving it to a desired region,  $Q$ , in the state space in a finite amount of time  $T$ . Physical systems have momentum, so state trajectories cannot always be turned instantaneously to another direction and therefore may exhibit overshoot. Further, control mode switches often result in transient behavior. So, by definition, Type II safety requires that any resulting overshoot or transients involved during the switching process and transition to  $Q$  will not compromise plant safety (will not leave the set  $S_{Dsafe}$ ). Therefore, the Type II safety definition is relative to the set  $Q$  and time  $T$ , bounding the duration of the transition.

We define the region  $Q$  to be that set of states at which the *emergency condition* (caused by the advanced system) to be resolved or alleviated. It need not be the final state region in the recovery process. That is, if the recovery process (activated by the RTA system) stops before the state reaches  $Q$ , then there exists the possibility that the state could transition outside of  $S_{Dsafe}$  if no further action is taken to prevent this from occurring. Once the state reaches  $Q$ , then it will be in a stable, safe region of attraction. The recovery process may then further act on the system to drive the state to a particular baseline operating region, depending on the specific application.

In practice,  $Q$  and  $T$  would be chosen to be consistent with overall system requirements and how best to respond to any performance degradations in switching to a less capable baseline system. For example, for inner-loop reversion, the set  $Q$  might represent the set of trim conditions for an aircraft in straight-and-level flight. In this case,  $T$  would represent the maximum amount of time the outer-loop could allow for such a transition before resuming command inputs to the inner-loop system. It is important to note that  $Q$  and  $T$  are not arbitrary, but intricately tied to the design of the reversionary system. The reversionary system does not have to be designed to take the overall feedback system from any point in the state space to any arbitrary region  $Q$ . Rather, the reversionary system is designed to specifically drive the state to a specifically designed region  $Q$  within a specifically designated time  $T$ .  $T$  itself also has design constraints. Even if the reversionary system has the capability to drive the state to a designated region  $Q$ , if it takes too long for it to do so, then unsafe conditions could result at other feedback levels and this would be an unacceptable reversionary system design. Last,  $Q$  and  $T$  are not unique and could be different for different operating conditions, or different for other reversionary system designs, depending on the desired behavior of the specific application.

Finally, Type III safety accounts for any behaviors of the advanced system between checks performed by the RTA component. From any point in the Type III safe region, the RTA switch can pass any commands from the advanced system to the plant for at least  $\tau$  seconds without exiting Type II safety. *Note that we define  $\tau$  to be a maximum value before safety can no longer be assured. Therefore, the update rate of the RTA monitor's checking algorithm must be no greater than the quantity  $\tau$ .* Otherwise, an output of the advanced system potentially exists that could drive the system to an unsafe state before the next RTA monitor update.

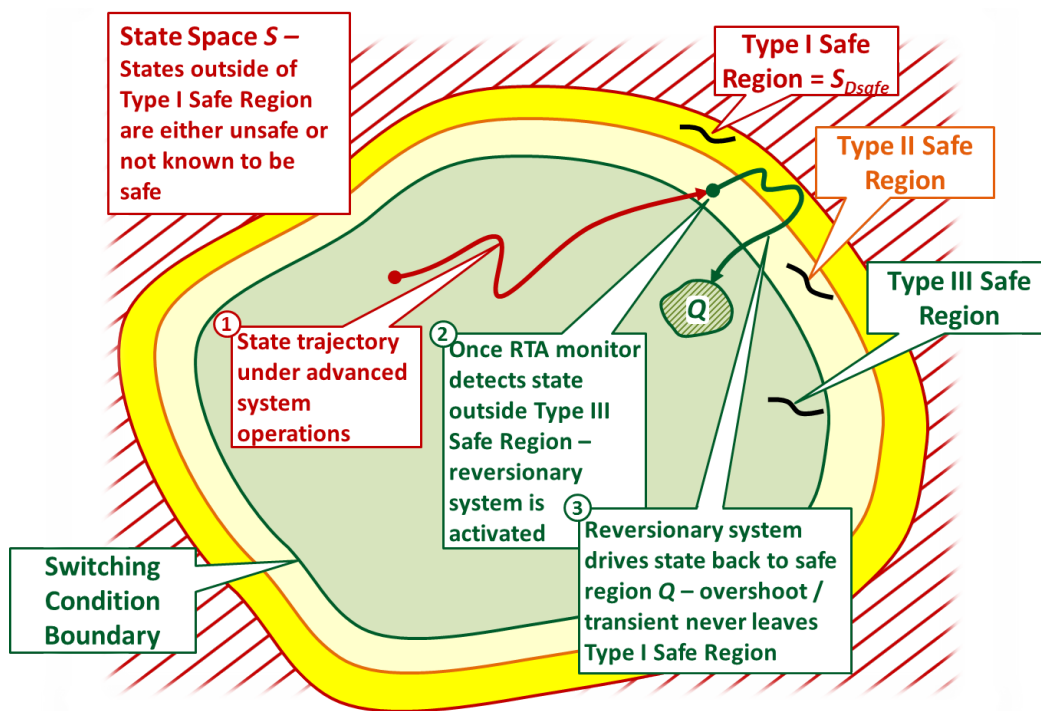
These definitions are perhaps more readily understood from the perspective of the RTA's decision logic. The RTA component must guarantee that the plant never leaves the Type I safe region, which ensures that the plant continues to operate safely. To guarantee this, every  $\tau$  seconds, it checks to see if the current state is inside the Type III with period  $\tau$  safe region. If the state is in this region, the RTA can allow the advanced system's commands to pass through to the plant. In the worst case, on its next check  $\tau$  seconds later, the plant's state will have transitioned out of the Type III region, but it will not have transitioned beyond the Type II safe region (Part b in the definition of Type III safety). In that case, the RTA component must switch to the reversionary controller. Because the plant is now in the Type II safe region, the resulting transient will never leave the Type I safe region (Part c of the Type II safety definition) and the reversionary system can successfully maintain safe control/management of the plant (Part b of the Type II safety definition). Lastly, note that by the definitions of Type I, II and III safety, recovery by the reversionary system is also not independent of the environmental conditions in which it is operating (Part a of the Type II safety definition). That is, by the definition of  $S_{Dsafe}$ , we do not allow the system to encounter hazardous environmental conditions within the time the state is transitioning to  $Q$  (the reversionary system is *rated* to correctly operate only within  $S_{Dsafe}$ , which defines bounds on environmental conditions).

We use the term switching condition boundary to refer to the surface of the Type III safe region. This constitutes the border between the Type III and Type II safe regions. This term emphasizes the fact that the RTA component switches to the reversionary system if the plant's state trajectory ever crosses that surface. At the control and guidance law feedback levels, which have very fast dynamics, the Type III boundary clearly defines the switching condition boundary. Here,  $\tau$  will be a comparatively small value and notionally  $\tau$  is the update rate of the RTA monitor, which should be at least as fast as the update rate of the advanced system. If the RTA monitor does not activate the switch to the reversionary system once the state has crossed the Type III boundary, then the state could potentially violate the Type I boundary at the next system update.

At the FMS and MPS levels, these subsystems are planning out the mission and vehicle paths over comparatively long horizons (tens of seconds to minutes and even longer). In this case, the maximum value for  $\tau$  may be quite large (on the order of minutes). We assume, however, that the update rates of the FMS and MPS blocks and their respective RTA monitors are updated at a much faster rate (1 to 5 Hz, for example) than the value for  $\tau$ . In this case, the checks that are performed by the RTA monitor may likely be able to determine an error has occurred long before there is any actual compromised safety. If so, there would be no reason to continue to operate the advanced system until the *last possible moment* (i.e., waiting for as long as  $\tau$  seconds) since it is known that the advanced system is producing incorrect solutions. In these cases, the RTA

systems are much more akin to *software integrity monitors* than safety monitors, and the switching condition would be determined by certain tests on the outputs or solutions offered by the advanced systems, rather than on the safety switching condition of the Type III boundary.

Exact procedures or methods to construct Type I, II and III boundaries have not been fully developed (e.g., through targeted simulations, or more formal analyses – see Subsections 2.7.2 and 2.7.3). However, formally defining these boundaries gives the necessary foundations to begin that process. Figure 8 illustrates the relationship between the Type I, II, and III safe regions for a 2-D slice though their respective multi-dimensional hypersurfaces in state space. The figure also illustrates a state trajectory under advanced system operations. When the RTA monitor first detects the trajectory has left the Type III region, it switches to reversionary system operation, which drives the state trajectory to the desired region,  $Q$ . Once the switch to the reversionary systems is performed, the resulting trajectory is entirely contained within the Type I safe region, and thus the safety of the plant is never compromised.



**Figure 8. Relationship between Type I, II, and III Safety Regions**

Note that under RTA protection, operation of the advanced system is limited to the Type II safe region; i.e., the advanced system's operating envelope is the Type II safe region. It will, of course, nominally operate only within the Type III region, but the above protocol allows the advanced system to drive the state to the Type II region between RTA updates before it is deactivated and operation is switched to the reversionary system.

The operating envelope of the reversionary system is the Type I safe region. Here too, however, it will nominally operate within the Type III safe region, but the above protocol allows for the state trajectory to enter into the Type I region during the transition from the RTA switch to the desired region  $Q$ . Therefore, nominally the set  $Q$  and the operating regions of all baseline

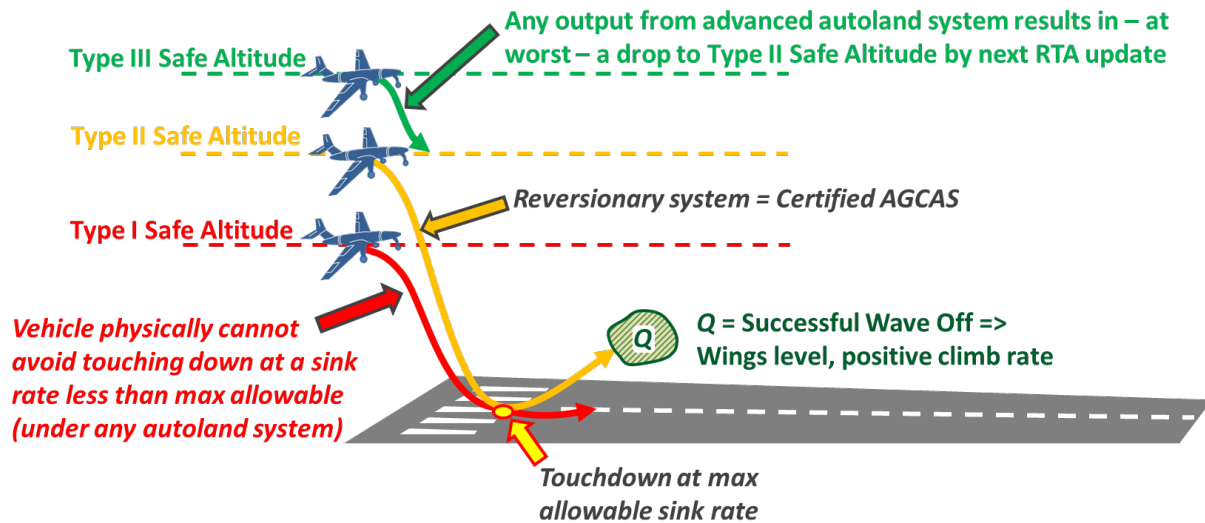
systems within the reversionary system's operating region will all lie within the Type III safe region. Although our definitions above do not disallow cases in which  $Q$  or the operating region of a baseline system lie outside of the Type III safe region, we have not considered such cases since this would imply reversionary system operation at or near the Type I safe region's border, contradicting typical safety margin protocols.

### 3.2.4 Illustrative Example

To illustrate these concepts, consider the challenge problem in the prior CPD program [Aiello 2010]. In that program, we developed an RTA system to protect a vehicle using an advanced autoland system. The reversionary system was a Lockheed-Martin developed certified automatic ground collision avoidance system (AGCAS) that would perform *wave off maneuvers* which would command the vehicle to go wings-level, pull up and then increase thrust to provide a positive climb rate. For the RTA switching condition, we developed a safety boundary (crudely equivalent to a Type II safe boundary) along the autoland approach corridor by executing thousands of simulations with varying initial state conditions. At each initial condition, we simulated the vehicle running the certified AGCAS and recorded whether the vehicle hit the ground, and if so, whether the sink rate was less than or greater than the maximum allowable sink rate magnitude. If it landed greater than the maximum allowable sink rate, then this could potentially cause landing gear or other vehicle damage, thus resulting in unsafe conditions. If the vehicle touched down at less than or equal to the maximum allowable sink rate magnitude, or was able to avoid the ground altogether, then that initial condition state was considered to be within the safe operating region.

Although we had not yet defined the Type I, II or III safety regions in the CPD program, we can use that demonstration problem as an illustrative example to highlight understanding of these safety definitions. Figure 9 illustrates the Type I, II and III safe boundaries for altitude alone. Note, however, there are several states that can influence whether the vehicle can successfully perform a wave off maneuver, including airspeed, altitude, AoA, vehicle attitude, current weight, landing gear mode, flap deflections, etc. We assume that in this example one or more of these other states are at particular values that will require a wave off maneuver if the altitude drops below its critical Type III boundary.

This figure shows that at the Type I safe altitude boundary, the vehicle physically cannot avoid touching down at a sink rate magnitude less than the maximum allowable magnitude. This is by the definition of the Type I safe boundary (i.e., the aircraft is at the edge of safety). At this point, no autoland or control system exists that can *soften* the landing any more than the maximum allowable, and in fact, poorly designed autoland/control systems could actually cause the vehicle to touchdown harder than the maximum allowable value. The Type I safe boundary is a function only of the physics of the plant. Note, however, that we are only considering a one dimensional *slice* in the multi-dimensional hyper-surface of the Type I boundary. For example, if the initial airspeed is faster, then the Type I safe altitude would lower in value, and conversely, for a slower airspeed the Type I safe altitude would increase in value. Likewise, changes in initial AoA, pitch attitude, weight, etc. would all influence the Type I safe altitude.



**Figure 9. Illustrative Example – Type I, II, and III Safety for Altitude State**

Next, Figure 9 shows the Type II safe altitude boundary. This altitude is determined by the ability of the reversionary AGCAS to successfully perform a wave off maneuver and drive the state to a desired region,  $Q$ , within a certain time  $T$ . The region  $Q$  is chiefly characterized by a wings-level, positive climb rate state, although other states, such as pitch angle and airspeed would also be defined (or acceptable ranges in these states would be defined for the region  $Q$ ). The Type II safe altitude boundary is the point at which the AGCAS just touches the aircraft down at the maximum acceptable sink rate before pitching the vehicle up to begin a positive climb rate. The Type II safe altitude is higher than the Type I safe altitude boundary because additional altitude is needed to allow for the vehicle to gain enough airspeed to begin going up in altitude once it just touches the runway (i.e., to be able to eventually achieve  $Q$ ). Any altitude higher than the Type II safe altitude would result in softer touchdowns. Further increases in altitude will eventually result in avoiding the ground altogether.

Last, Figure 9 shows the Type III safe altitude boundary. At this altitude, any commands generated by the advanced autoland system will result in, at most, a drop in altitude to the Type II safe boundary. Typical update rates of autoland systems will be approximately 1 Hz, which implies that the Type II and Type III altitudes will be comparatively close together because the change in altitude a typical aircraft can achieve in one second will be relatively small, especially at low airspeeds during final approach. The autoland RTA monitor will command the switch to the AGCAS somewhere between the Type III and Type II safe altitude boundaries. Again, other states will influence where the switch is activated. For example, in the time between RTA monitor updates, not only can the vehicle drop in altitude, but it will also fly a certain distance downrange as well. If the vehicle is far down over the runway, there will be a point at which the downrange state becomes more critical than the altitude state and the RTA monitor will trigger a switch based on the Type III downrange boundary to ensure that, as the AGCAS is performing the wave off maneuver, it does not run out of clear runway.

Again, this illustrative example only focused on the altitude state and it is assumed that a combination of other states require a wave off maneuver. In the CPD program, through

simulation studies we constructed a safe-to-fly corridor mapping for all the states included in the demonstration problem. As long as the vehicle states remained within this corridor, there were no Type I, II, or III safe altitude boundaries and the RTA monitor would allow the advanced autoland system to fly the vehicle all the way to the runway with an acceptable touchdown sink rate.

### 3.3 Reversionary System Description

#### 3.3.1 Reversionary System Composition

Whatever approach is taken to provide runtime protection to systems with advanced, uncertifiable components, one of the main objectives is to not severely limit or restrict the operational envelope of the advanced system. Again, in our approach, the advanced system's operating envelope is limited to the Type II safe region. Therefore, the reversionary system should be designed such that this region is as close as possible to the Type I safe region. This can be accomplished if the reversionary system is designed to smoothly take over control when needed, minimizing transients arising from the control mode switch and having the control power to minimize state trajectory overshoots.

Also, in our approach, the reversionary system's operating envelope is actually required to be larger than the advanced system's operating envelope, since it must cover the Type I safe region even though it presumably has less performance capabilities than the advanced system. To accomplish this, we expand on the simplex framework idea (recall, discussed in Appendix A), and define the reversionary system to be comprised of the following:

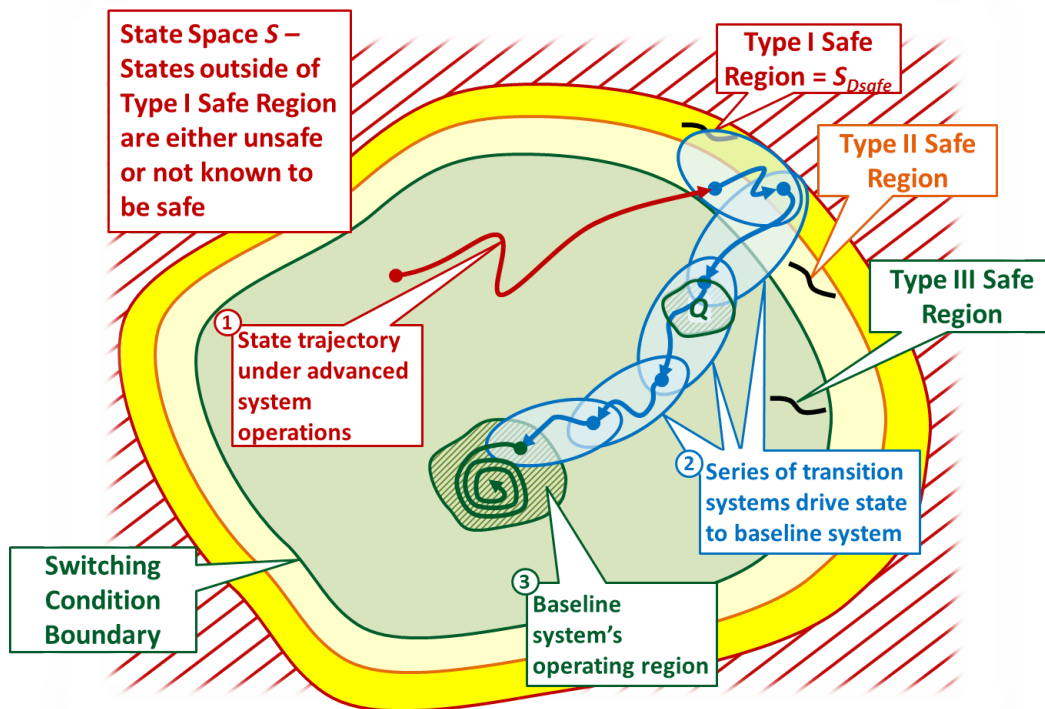
- a. Baseline systems: The reversionary system is required to have at least one baseline system, although it may have more than one depending on how the baseline systems are defined and how they cover the operating envelope of the reversionary system. The baseline systems may have much smaller operating envelopes than the advanced system, typically with the objective of ending the mission and safely returning the platform *back to base* (or return to a *safe state* or shutoff condition). Due to the simplicity of the baseline systems, they can be certified to the required criticality level at design time. Ideally, these systems are typically obtained from prior fully certified designs. The definition of the baseline systems here is equivalent to the baseline controller in the simplex framework, although we now allow for more than one baseline system, if necessary.

Note that in our prior work we only considered the reversionary system to be composed of one baseline system, thus the operation of the advanced system was, by default, limited to those regions where the single baseline system would be able to take control. We now allow for multiple baseline systems as well as transition systems that expand the operating envelope, discussed next.

- b. Transition systems: We expand the idea of the safety controller function in the simplex framework here to define transition systems. As above, we allow for more than one transition system, if necessary, and there may be several depending on the particular design. There are two main functions of the transition systems:



- i. The first transition system immediately takes control once the control mode switch is activated by the RTA monitor. This is critical for inner-loop control switching, but may also be an issue for outer-loop applications. This implies that the transition controller prevents integrator windup (prevents actuation saturation, for example), matches integrator values in some manner, transitions commands from the advance system to the baseline system in some manner (a linear progression, for example), to minimize system transients. The design of this first transition system may be generalized to some extent, but will have elements that are application specific depending on the system dynamics.
- ii. Like the safety controller concept in the simplex framework, the other main function of the transition systems is to drive the state from an ensuing unsafe condition to the operating region of a designated or chosen baseline system in a safe and stable manner. Therefore, after the first transition system has safely taken over control, there may potentially be a series of follow-on transition systems that are used to drive the system state to a baseline system's operating region. Transition systems' operating regions must *overlap* and they must each have the control power to drive the system state to the next transition system in the sequence so that the state is eventually driven to a baseline system's operating region. This is illustrated in Figure 10.



**Figure 10. The Role of the Transition and Baseline Systems in the Reversionary System**

How many transition and baseline systems are required will depend on the coverage each system provides relative to the reversionary system's operating region (again the Type I safe region) as well as the design objectives of the recovery actions. Taken together, the transition and baseline systems define the reversionary system. There needs to be enough transition systems such that a transition system is always available that can take over from the advanced system anywhere in the operating region where the RTA monitor can command a switch to the reversionary system

(the light yellow region in Figure 10 – between the Type III and Type II boundaries). There then need to be enough transition systems with overlapping operating regions to drive the state to a baseline system's operating region. The construction of the transition systems is design specific and depends on the objectives of the recovery procedures. Depending on the particular application, there may be no need for any transition systems if the baseline system can appropriately cover the Type I safe region. Most importantly, all baseline systems and all transition systems are required to be certified to the appropriate DAL at design time since these elements provide the certified recovery actions, protecting the overall system from any unforeseen errors in the advanced system, which is certified only to a lower DAL.

Also, note in Figure 10 that the desired region  $Q$  is illustrated to be *in the path of* the series of transitions systems. Here too, the relationship between  $Q$  and the transition and baseline systems is design specific. In the figure,  $Q$  represents the region of initial states for the third transition system (or, region of attraction for the second transition system). Recall, the set  $Q$  is defined as that region in state space in which the *emergency condition* (caused by the advanced system) is resolved. So, although there is a region of attraction for each transition system, the set  $Q$  is not necessarily the region of attraction of the first transition system (although it could be, as is the case in the illustrative example presented below). If the recovery procedures are stopped at the equilibrium of the first transition system, the state may still be in an unacceptable region with unacceptable safety margin. In that case, the recovery action must continue through more transition systems until the desired region  $Q$  is achieved. The region  $Q$  could be the operating region of a chosen baseline system, or there may be more transition procedures before the state reaches the baseline operation region. The relationship between  $Q$  and the transition and baseline systems is left general here and is specific to each system design.

### 3.3.2 Illustrative Example Continued

Returning to the example given in Subsection 3.2.4, Figure 11 depicts a possible reversionary system design that contains two transition guidance systems and two baseline guidance systems.

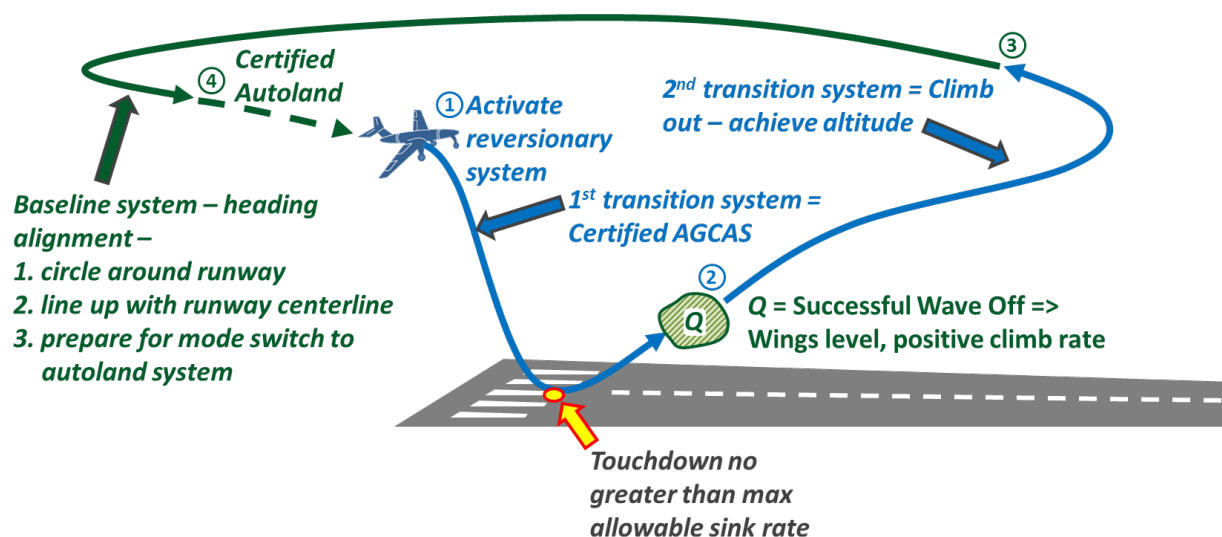


Figure 11. Illustrative Example Depicting Transition and Baseline Systems

This figure illustrates the following:

1. The RTA monitor determines that the system state is such that if the advanced autoland system were allowed to continue to operate, then a touchdown sink rate greater than the maximum allowable sink rate would result, causing potential equipment damage. The advanced autoland system is therefore no longer trusted and is permanently deactivated. The reversionary system is then activated, switching control to the first transition system, which is shown to be the certified AGCAS. It recovers the platform by performing a wave off maneuver and achieving the desired state in the region  $Q$ , again characterized by wings level with a positive climb rate.
2. The second transition system then takes over, performing a climb out maneuver, clearing the runway and gaining the proper altitude to transition to the first baseline system.
3. The first baseline system then takes over, circling around the runway through a predetermined flight corridor to satisfy terminal area air traffic management requirements. It then aligns the vehicle with the runway centerline and properly commands the airspeed and vehicle attitude to transition to the second baseline system.
4. Last, the second baseline system takes over, which is a certified, trusted autoland system. This system successfully lands the vehicle within its acceptable state limits.

This example shows a reversionary system design with multiple transition and baseline components. We will present more examples in the following chapters of reversionary system designs and their specific component functionalities when we address the program's challenge problem demonstrations.

### 3.3.3 Reversionary System Shadow Mode Operations

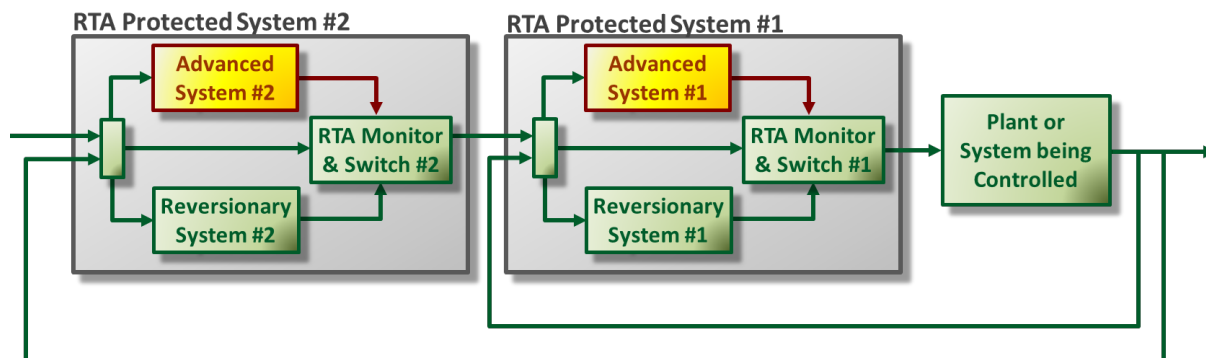
In most applications, while the advanced system is active, it is recommended that the reversionary system also be running in *shadow mode* in which its output is not used, but is calculated. Running in shadow mode has certain advantages, such as keeping track of the current values of all control effector deflections, active integrator wind-up prevention, etc. This helps in minimizing any transients if control needs to switch to the reversionary system. Therefore, presumably, it will be the first transition controller, or the set of first transition controllers that may need to run in shadow mode since these will be the systems to first take over operation from the advanced system. Alternatively, the complete reversionary system may need to be always running in shadow mode, continuously driving a *shadow state* to an appropriate baseline system's operating region.

The practice of running in shadow mode is common in many applications, such as guidance and control systems for multistage launch vehicles. Commonly, at all phases of the ascent to orbit, the current lowest stage will be in control of the vehicle. At staging, the guidance and control systems of the next most upper stage need to come online after running in shadow mode [Schierman 2012].

## 3.4 Interacting RTA Protected Systems

Recall from Figure 5 that we will be investigating runtime protected systems at any and all feedback levels in an aircraft's nested feedback architecture. This results in interacting RTA

protected systems, which requires us to address additional considerations not present in a system with only one runtime protected system. Note that in Figure 1 we put placeholders for either upstream or downstream blocks. Let us now consider an additional upstream RTA protected block, as shown in Figure 12.



**Figure 12. Interacting RTA Protected Systems**

In this figure, the upstream RTA protected system No. 2 delivers inputs to the downstream RTA protected system No. 1, which then delivers commands to the plant. However, plant output can be fed back to both RTA protected systems; thus the downstream system can influence the upstream system indirectly. Although not shown in the figure, the output of RTA protected system No. 1 may be directly fed back to RTA protected system No. 2, depending on particular design requirements.

### 3.4.1 RTA Monitor Checks

Because overall system safety is now a function of two RTA protected systems, each RTA monitor must perform additional checks. We present here the main checks that each RTA monitor performs in a system with interacting, nested runtime protected systems.

1. Safety check: As already discussed, at each feedback level the RTA monitor checks the system state to determine if it has left the Type III safe region. If so, the advanced system is shut down and control is switched to the reversionary system.
2. Output/environment check: The RTA monitor also checks that the outputs generated by the advanced system do not violate any constraints imposed on the downstream system or plant. Such constraints may include maximum/minimum values, limits on rates of change of outputs, frequency content or other design specific statistical measures. Frequency content constraints or statistical measures are typically performed on running windows of past and current output data. These checks fundamentally determine whether the downstream feedback level is being *over-driven* such that it cannot physically perform what it is being commanded to do (e.g., commanded altitude change requires a climb rate greater than the vehicle can achieve, or frequency content of the output is higher than the break frequency of the downstream block such that it cannot properly follow the delivered commands). At the higher level mission planning feedback loop there may be checks to determine if its output is driving the platform into an environment where the physical system was not designed or intended to operate (e.g. high winds causing uncontrollable flight, or smoke/dust conditions

that blind optical sensors). There also may be constraints on the order of certain mode switching logic, at the mission planning or flight management levels, for example. Here, if mode changes are commanded in an out-of-order sequence, then this would violate dictated constraints

Again, the details of how these checks are performed are specific to the particular system design. It is presumed that a properly designed advanced system will not violate such constraints. Therefore, if one or more output constraint violations are detected by the RTA monitor, the advanced system is shut down and control is switched to the reversionary system. Note here, such constraint violations may not necessarily affect safety at the current feedback level, but can drive the next downstream system into an unsafe state; and because of feedback, the entire nested feedback system may eventually become unsafe.

3. Performance check: The RTA monitor also checks that the advanced system responses adhere to its specified performance requirements. Such specifications are typically constructed during the design and analysis of the advanced system. These may include transient response specifications, such as rise time for commanded step inputs, or settling times for disturbance rejection measures. Steady state norm bounds on tracking errors are another example of a common performance measure. Performance measures for higher level flight management or mission planning elements may include some measure of mission progress, (e.g., is the vehicle arriving at a designated objective at the correct time?). These measures will be application specific for the particular advanced system design, but should be included as a part of the RTA monitor's checking.

As with output constraint violations, poor performance does not necessarily imply the system is operating in an unsafe state at the current feedback level. However, since the RTA protected block is no longer acting in isolation, poor performance at the current feedback level could cause safety hazard conditions at downstream levels, or due to feedback, at upstream levels. For example, consider that a vehicle is flying toward a no-fly zone and the flight management level commands the vehicle to avoid the zone by following a set of waypoints or a particular trajectory through space. In turn, the guidance level commands the advanced inner-loop control system to follow that trajectory or series of waypoints. However, if the tracking performance of the inner-loop system is poor, the vehicle may enter into the no-fly zone due to, for example, a large crosstrack error. Therefore, even though no safety violation is detected at the inner-loop level, its poor performance results in a safety violation at the flight management and guidance levels. Furthermore, even if no safety violations at other levels results from poor performance at the current level, the fact that the advanced system cannot achieve stated performance specifications indicates that some error is present in the code or algorithm design, and per the requirements from DO-178B/C, flight critical software that does not work correctly through all traces of certification artifacts should be considered untrusted.

We assume here that there may be a need for some margin in the performance checks, just as we built margin in the safety checks through the Type I, II and III framework. However, detailed analysis of how such margins should be constructed for performance checking has not been done in this project and we leave that question to be addressed in follow on efforts. See Chapter 13 for recommended further studies.

4. Input/environment check: The RTA monitor should also check that inputs or commands to the current feedback level do not violate any constraints imposed on the inputs. Just as with the output check, such constraints may include maximum/minimum values, rate limits, frequency content, statistical measures, mode logic, operating environment constraints, etc.

It can be argued that this check is redundant if the immediate upstream system is an RTA protected block, in which case its RTA monitor would have performed its own output check, ensuring that only correct outputs are delivered to the current feedback level, maintaining safe operation. This would also be the case if the upstream system was a design time certified block, in which it is presumed to always deliver correct commands downstream. However, we include this check so that no certification restrictions are placed on the upstream block. This may allow for more design freedom during experimental testing stages. That is, each runtime protected block makes no assumptions on the certification or safety assurance level of any other system that can influence its operation.

Related to checking input validity is checking that the operating environment or operating conditions fall within specified ranges. It is assumed that this check would be performed using available sensor information. The vehicle itself may be operating correctly, but if it is driven into environmental conditions for which it was not designed, then this indicates errors in the software that drives the vehicle state into such unsafe conditions.

5. System hardware health check: as discussed previously, the RTA monitor may be required to interact with a system health monitor that indicates whether the information being sent to the monitor and other system components is valid and whether there are any control effector failures or other hardware damage that could be causing anomalous behavior. This knowledge is critical in determining if faulted conditions are due to hardware problems or due to software faults. If the former, then the RTA system should allow operation of the advanced code to continue, since no faults are indicated in the advanced system under such conditions.

### **3.4.2 RTA Monitor Viewed as an Assume-Guarantee Contract Checker**

The process of checking input constraints and output performance is very similar to the compositional verification approach that derives assume-guarantee (A-G) contracts over a system of subcomponents that make up the complete system. We have explored the relationship between the RTA checks discussed above and the compositional reasoning approach. In the following chapters, we will develop the A-G contracts at each feedback level and use those contracts to then form the specific RTA checks at each feedback level.

Rockwell Collins is developing an automated assume-guarantee verification tool entitled, “Assume Guarantee REasoning Environment” (AGREE). The following is reproduced from [Cofer 2012]:

“Assume-guarantee contracts provide an appropriate mechanism for capturing the information needed from other modeling domains to reason about system-level properties. In this formulation, guarantees correspond to the component requirements. These guarantees are verified separately as part of the component development process, either by formal or traditional means. Assumptions correspond to the environmental

constraints that were used in verifying the component requirements. For formally verified components, they are the assertions or invariants on the component inputs that were used in the proof process. A contract specifies precisely the information that is needed to reason about the component's interaction with other parts of the system. Furthermore, contract mechanism supports a hierarchical decomposition of verification process that follows the natural hierarchy in the system model.”

In [Cofer 2011] it states:

“...for each component within the architecture, we define a set of assumptions that the component expects to always be true of the external environment and a set of guarantees that the component will always satisfy. In other words, the guarantees are the component functional requirements, and the assumptions define the context in which we can use the component. Together, the assumptions and guarantees for a component form a contract for that component. If a component is composite (that is, defined by other subcomponents), then we use the assumptions and guarantees of the subcomponents, as well as facts known about the architecture by the instantiation of patterns, to prove that the component satisfies its guarantees.”

From ARP 4754A, one foundational concept for assessing system safety is to check for correctness of operation with respect to the defined intended function of the component. Certainly this should be validated at design time. However, in addition, using RTA monitors as A-G contract checkers also validates that intended functional requirements are always met during runtime, further increasing trust in system safety and correctness.

### 3.4.3 The Need for a Global RTA Manager

For systems with multiple interacting RTA systems, an overall system or global RTA manager will be required to coordinate control mode switching processes or pass critical information to all feedback levels affected by the switch to a reversionary system at one particular level. We augment the system shown in Figure 12 with an RTA manager block, which lies outside the feedback loops of the runtime protected systems. This is shown in Figure 13.

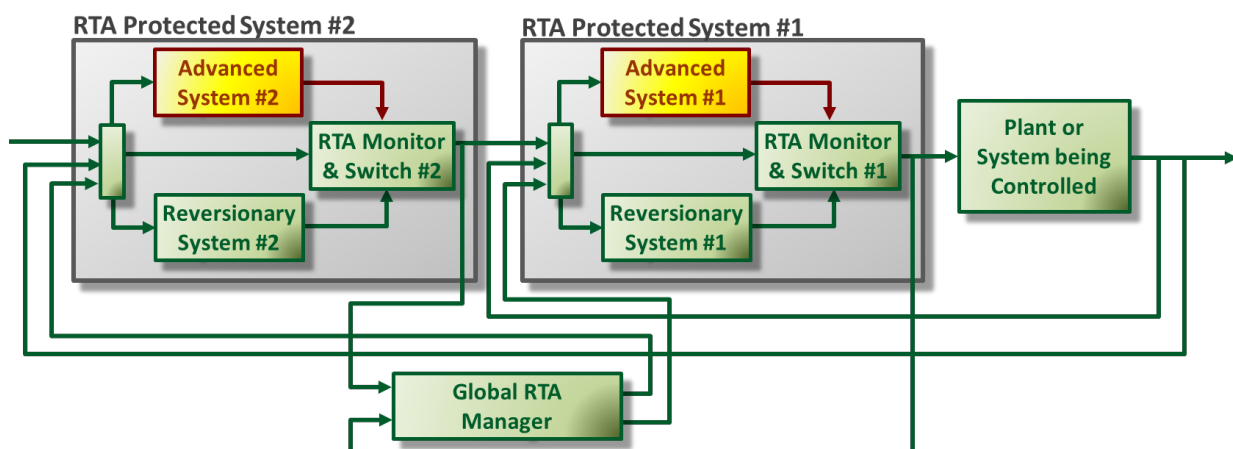


Figure 13. Global RTA Manager for Multiple Interacting Runtime Protected Systems

This figure shows that the output of each RTA protected block is delivered to the global RTA manager block, and the manager block generated output that is subsequently fed back to each RTA protected block. Since there is an integrated passing of information, the design of each protected block will need to have input/output compatibility with the RTA manager block.

Some of the main functions of this RTA manager are discussed below. Note that, as with all other components in the RTA system, the RTA manager must be fully certified to the appropriate DAL at design time.

#### **RTA Manager Communicates Control Mode Status and Current Performance Capabilities**

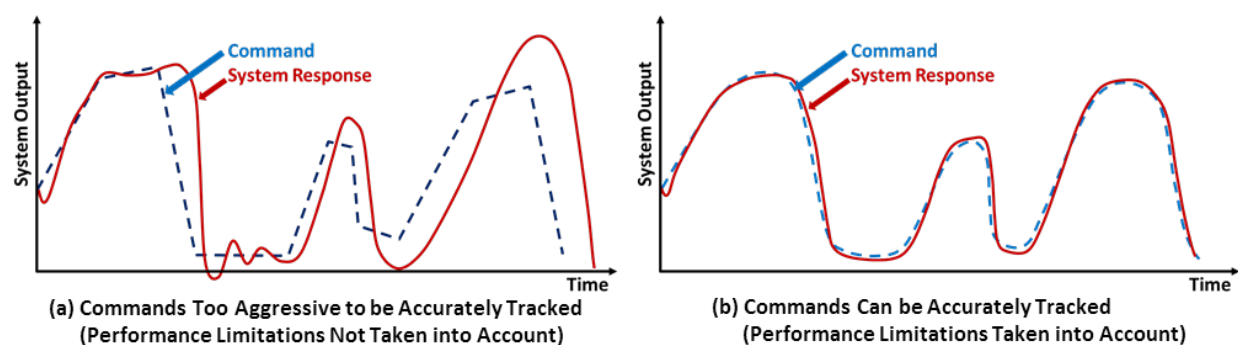
One main function of the RTA manager is to communicate the mode status (advanced or reversionary mode) of each feedback level to all other feedback levels. There may be a number of reasons for doing this, but one main reason is management/coordination of performance capabilities for the interacting feedback levels. We previously discussed that one of the main checks the RTA monitor executes is whether the advanced system is achieving its required performance. Assume that each level of the overall feedback system has a defined set of minimum performance requirements (however defined, specific to the particular design). For one particular level, define that set as  $P_{R\_min}$ . That feedback level must always be able to achieve  $P_{R\_min}$  because all other levels (both upstream and downstream) were designed assuming that level is operating at least to the performance of  $P_{R\_min}$ . By definition, the reversionary system for that level is design-time certified to always be able to achieve  $P_{R\_min}$ . Next, define the advanced system's minimum set of performance requirements as  $P_{A\_min}$ . There are two cases to consider here:

- i.  $P_{A\_min} = P_{R\_min}$ : in this case, if the RTA monitor commands a switch to the reversionary system, there is no change in the minimum performance requirements, and the RTA manager does not necessarily need to communicate that the switch to the reversionary system occurred.
- ii.  $P_{A\_min} > P_{R\_min}$ : in this case the advanced system was designed to have greater minimum performance than the original minimum requirements. If the RTA monitor detects that  $P_{A\_min}$  is not being achieved, then this indicates something is wrong with the advanced system and it can no longer be trusted. The RTA system then commands a switch to the reversionary system. Now, if the reversionary system is only guaranteed to be able to achieve  $P_{R\_min}$ , then the guaranteed performance capabilities of that feedback level have reduced. Although it seems counterintuitive to purposefully reduce performance capabilities, it should be done since the advanced system can no longer be trusted. However, the mode switch to the reversionary system should be communicated to the other upstream or downstream feedback levels in the event those levels were assuming the greater performance capabilities of the advanced system were available. This communication of mode status ensures that no feedback level *overdrives* another feedback level and each level has knowledge of the performance capabilities of all other levels.



Case ii is clearly a more complicated, more integrated architecture. Case i will most likely be the intended architecture for more immediate future advanced aerospace systems, and we will adopt that approach here. That is, for all feedback levels, let us assume their respective reversionary systems can achieve a minimum standard of performance that is assumed by all other feedback levels.

Knowledge of a system or subsystem's performance capabilities is important and can have safety implications. Figure 14 provides an illustrative example. Case (a) in the figure shows a system being overdriven with commands that are too aggressive with rapid changes and frequency content that the system is unable to accurately follow. Especially for nonlinear systems, growing tracking errors can quickly escalate to unacceptable performance and potentially unstable conditions. Case (b) in the figure shows commands that take into account the performance limitations of the system and provide smooth, less aggressive commands. Although the commands generally follow the same trends and magnitudes as in case (a), the tracking errors are kept to small values as the system has the performance capability to follow the commands.



**Figure 14. Tracking Performance Reduction due to Overdriving System**

As long as the reversionary system's minimum performance capabilities can match the required system performance, then scenarios like that illustrated in Figure 14 (a) will be avoided no matter what the reversionary status of each level is at the current time. If the advanced systems are active and are capable of achieving greater performance than that minimum standard, then the overall system may be able to take advantage of those greater capabilities, but no assumptions are made that more performance is guaranteed available.

#### **RTA Manager Coordinates Switching Protocols**

There may be specific switching protocols depending on the application and the feedback pathways. These too would be coordinated by the RTA manager. In general we see three classes of switching protocols:

- i. If any feedback level's RTA system activates reversionary mode, then all levels activate reversionary mode.
- ii. If any feedback level's RTA activates reversionary mode, then only downstream levels activate reversionary mode.

- iii. If any feedback level's RTA activates reversionary mode, all other levels continue to operate in advanced mode until their respective RTA systems activate their specific level's reversionary mode.

We consider case i) to be the most conservative, safest protocol since feedback interconnects upstream and downstream components. Therefore, if problems arise at any one level, such problems will be propagated throughout the complete feedback system. Options ii) and iii) above are related to the idea of *graceful degradation* in which advanced components are isolated as much as possible, and taken offline only as a last resort. Such protocols may require fault isolation algorithms to determine specifically what is causing the anomalous behavior (i.e., is the vehicle flying into a no-fly zone because the advanced inner-loop controller cannot accurately follow the commanded path, or is the commanded path or the plan itself in error due to faults in the advanced components of the mission planning, flight management or guidance systems?). In any event, graceful degradation switching protocols are attractive in that they take advantage of the advanced components as much as possible. We will discuss this topic further in later chapters in this report as we focus on the complete interacting system.

### **3.5 Additional RTA System Considerations**

#### **3.5.1 Switching Protocol**

In this program, our main application is safety critical unmanned aerospace systems and we adopt the following switching protocol: if the RTA monitor determines that switching to the reversionary system is required, then the system remains in reversionary mode for the rest of the mission until successful, safe recovery of the vehicle is accomplished. Our viewpoint here is that if mode reversion is necessary, then some algorithm design or software coding error is present in the advanced system that caused the mode reversion; therefore it can no longer be trusted to safely operate.

#### **3.5.2 Complex versus Simple Boundary Checks**

Recall from Section 3.2 that we defined the Type III safety boundary as the control mode switching condition. Again, this may be a complex hypersurface and a function of many interdependent system states requiring some type of query algorithm or other distance measure (e.g., signed distance functions, as discussed in [Schierman 2014] to determine if the system has indeed crossed this boundary, requiring a control mode switch to the reversionary system. We categorize this as *complex boundary checks* in the RTA monitor.

However, there may be some states or critical parameters of the system that can be checked in isolation from the complex boundary checks (or, in addition, *need* to be checked) that have very simple relations, such as constant upper or lower bounds, or bounds that are only a function of one or two other conditions or states, such as Mach number or altitude, *but have no other dependencies on any other system states*. Again, we have observed that some systems, such as aircraft turbofan engine have clearly defined, finite sets of physical parameter limits that make developing a safety monitor straightforward. We propose here that it may be more efficient to isolate the checking of parameters that have simple relationships for their upper and/or lower bounds from the rest of the complex boundary checks. If there exist certain critical parameters or states that have clearly defined physical or derived limits that are directly measurable (or readily reconstructed through estimation techniques), then monitoring these parameters becomes

straightforward. We categorize this as *simple boundary checks* in the RTA monitor. If their limits are breached (within some defined margins) the RTA monitor commands a control mode switch to the reversionary system.

This approach takes advantage of any simplicity in the system that provides straightforward switching decisions. In some complex systems there may be no straightforward checks, therefore there would be no simple boundary checks. In other systems (turbofan engine inner-loop control), system safety may be completely checked by simple relations, and there would be no need for complex, hypersurface boundary checks.

### 3.5.3 RTA Frameworks Operating with Other System Code

The avionics on aircraft can be comprised of quite complex systems and the feedback elements we address here make up only a part of the whole set of onboard electronic systems. In some cases, there may be additional subsystems that interact with RTA protected blocks that are neither a part of the advanced block, the reversionary block, or the RTA monitor block and may reside outside of the feedback loops altogether. If such code resides *outside* of the protection afforded by the runtime monitoring, it will, of course, be required to be certified to the appropriate level. Such code may simply be supporting code performing certain intermediate functions or providing certain intermediate information used by the RTA protected systems. Or, this code may operate across the feedback levels and may have *authority* over any or all feedback levels. We will discuss this case when investigating the interactions between RTA architectures and certified collision avoidance systems. Blocks residing outside the RTA protected feedback levels is depicted in Figure 15.

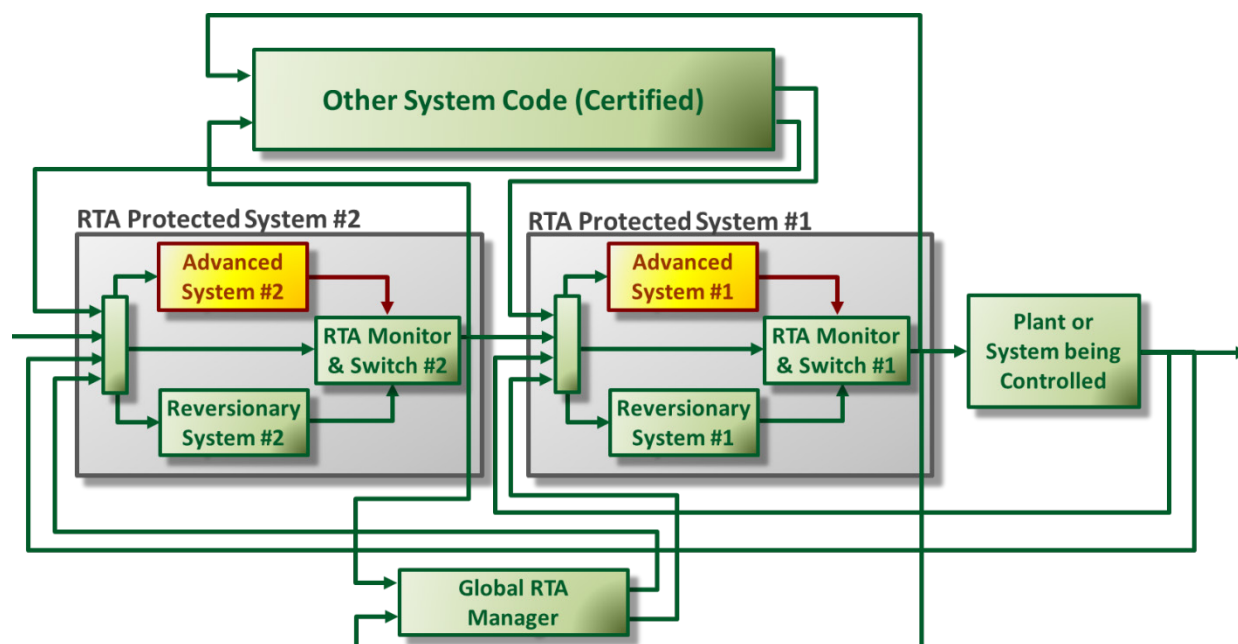


Figure 15. RTA Protected Loops Interacting with Other System Code

#### **3.5.4 Certifiable Code Residing within Advanced Systems**

There may also be subsystems within the advanced block that are simple enough to be V&V'd (at least in isolation) at design time. However, if they are a part of the advanced system, then we simply include such code within the advanced system block since certification is established in the context of the whole block operating within the complete aircraft system. Such code may be comprised of some type of supporting utility functions that perform intermediate calculations and provide intermediate inputs to the truly advanced algorithm within the advanced system block. This will be shown to be the case with the advanced inner-loop controller for the morphing wing system. Certain sub-blocks simply perform coordinate transformations, for example, that are used in the advanced, adaptive controller. Rather than isolate these blocks of simpler code from the adaptation function, the entire set of sub-blocks are defined to reside within the advanced inner-loop controller.

Furthermore, RTA can be a form of post deployment V&V, allowing advanced systems that have not been tested with enough sufficiency to produce the proper design assurance prior to operational status to be fielded sooner, but in limited capacity. This approach allows data to be collected during initial testing stages that can then be provided to certification review boards as further evidence of testing, increasing the confidence in the advanced system. This approach is in line with the DoD TEVV of Autonomy Strategy, advocating *progressive sequential testing* within the initial operational phases of a system lifecycle [TEVV 2015].

## 4 RTA Applied to the Selected Challenge Problem

Up to now we have presented the RTA protected system framework in general terms. This chapter now focuses on applying that framework to the challenge problem first discussed in Section 2.5 and Figure 5. In this chapter we will discuss key considerations for the overall feedback architecture for the challenge problem.

### 4.1 Challenge Problem Selection Objectives

Future systems of interest to the Air Force (fielded 10 to 20 years in the future) were a primary objective in considering challenge problem models. This included hybrid cyber-physical SoSs involving several different platforms working in some distributed or decentralized manner with intelligent autonomy. Therefore, for this program we focused on complex, highly adaptive, intelligent, nondeterministic control, guidance, flight management and mission planning systems for both single UAS platforms and cooperative control of multiple UAS platforms. This will begin to lay the groundwork for scaling up the complexity of RTA approaches to more complex SoSs in follow-on programs.

The RTA system should be demonstrated to be able to address and manage unpredictable behavior that may arise in these advanced elements due to unforeseen environmental conditions or changing mission scenarios, for example. Because of this, we first investigated bird flocking models (the dynamics of which are related to fleets of UASs performing a cooperative mission). Appendix B presents our investigation into several variants of flocking models common in the literature. The objective of this investigation was to develop flocking models that could be used to demonstrate adverse emergent behavior of complex interacting systems. However, the models we developed were too simple to be capable of demonstrating the complexities of emergent behavior, so we did not pursue this topic further. The results of this work may prove useful in follow on programs in which the study of emergent behaviors for SoSs is the main focus.

We then turned our focus to modeling UAS dynamics to begin construction of a fleet of UASs for the study of RTA protected systems for this application. This defined our challenge problem.

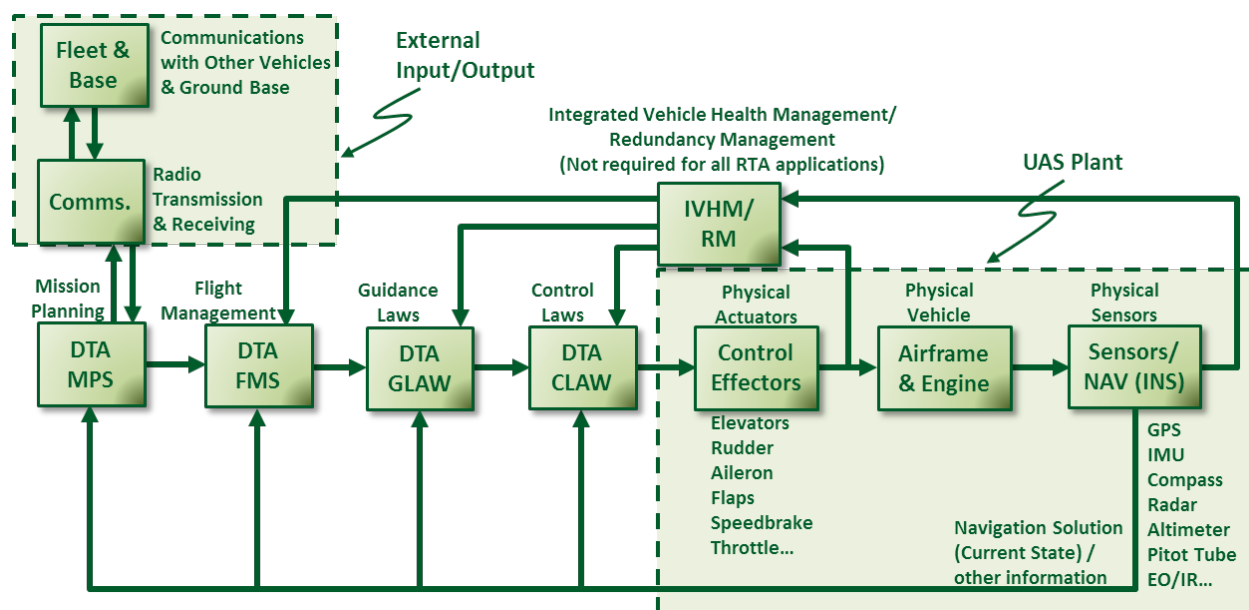
### 4.2 Overview of Selected Challenge Problem

1. Inner-loop control RTA: morphing wing application. The objective of the Option I program was to investigate the inner-loop control level. The Option I effort has been completed as a separate investigation and the results of that effort are presented in the next chapter of this report along with experimental results demonstrating the RTA system reversion. In one of our past AFRL-funded projects, we developed an adaptive/learning inner-loop controller for a rapidly shape-reconfiguring wing implemented on a small UAS vehicle. We chose this as the inner-loop challenge problem because it lends itself to clearly defining a baseline controller with a severely restricted operating envelope compared to the advanced controller. The baseline controller (or set of baseline controllers) was defined as having a valid operating region only around one wing shape configuration. In contrast, the advanced controller rapidly adapts to all different wing shape configurations and therefore, by default, has a much larger operating envelope. We developed an approach for constructing a series of transition controllers that takes operation from any wing configuration to the appropriate baseline configuration. In this work, we used a detailed 6-DOF model of the morphing wing UAS model. This dynamic model is presented in Appendix C.

2. Outer-loop guidance RTA: advanced trajectory following. We considered the advanced element of the guidance system as a continuous trajectory following system with adaptation capabilities. Such a guidance system was developed for a recently completed AFRL program entitled Multi-Vehicle Unmanned Aircraft Systems Sense And Avoid (MUSAA). This development and experimental results are presented in [Cooper 2014]. One subtask in the MUSAA program was to integrate the L1-adaptive control methodology [Gregory 2009], [Kaminer 2010] into the various modes of the guidance system as a means of adding robustness to uncertain environmental factors such as winds/turbulence. Because it is adaptive, this would certainly be a valid candidate for RTA protection. In the MUSAA program, L1 was integrated into a nominal guidance mode, but not into the continuous trajectory following guidance mode, (which is active only during collision avoidance maneuvering) due to funding limitations. We have not pursued actual inclusion of the L1 controller into any simulation demonstrations in this program. However, the guidance level is integrally involved in the investigation of multi-loop level RTA systems and we assume the continuous trajectory following guidance law contains some type of adaptation, such as L1, and we pursued RTA framework design considerations for the GLAW feedback level. For the outer loop guidance law and higher level feedback loops, we developed a 3-DOF simulation environment for multi-vehicle UAS experimental studies. An overview of UAS platforms and the 3-DOF dynamic model used in this program is given in Appendix D.
3. Mission planning and flight management RTA: multiple UASs in cooperative/decentralized control. Much of the project's focus has been on development of the mission planning and flight management levels as we have not addressed these levels in any past projects. The mission with multiple UASs in cooperative/decentralized control is characterized as an advanced, networked SoSs, possessing autonomy and intelligence with cooperative control and learning elements for dynamic mission operations. There may be optimal/dynamic resource management or asset allocation within groups of UASs or among differing teams of both manned and unmanned systems. The teams may be homogeneous or heterogeneous with mixed performance capabilities, and may sometimes have human operators in the decision making process, which further adds to uncertainty and potentially changing mission scenarios. Development of mission planning and flight management systems for fleets of UASs readily aligns with AFRL and DoD goals of enabling certification of levels of autonomy and can be integrated into such ongoing pursuits in this area, [DoD 2015], [OSD 2005].
4. Interacting RTA protected feedback levels. From our work involving the different feedback levels, it became apparent that the study of RTA protected systems cannot be performed exclusive of the effects and influences of the other upstream or downstream elements in the overall feedback system. Therefore, related to the three prior challenge problems, in the final year of the program we explored the interactions and interconnectivity of the multiple RTA systems in the nested feedback architecture. Here we explored the implications of graceful degradation switching protocols and also developed an overall safety case argument for the complete feedback system including all feedback levels. We were able to demonstrate that fault mitigation provided by the RTA systems ensures overall system safety during runtime.

### 4.3 Feedback System for Challenge Problem Ownship UAS

Recall that we first presented the overall feedback system for the challenge problem in Section 2.5 and in Figure 5. We noted that the block diagram in Figure 5 was a simplified representation. Let us expand out the elements of that diagram here. Figure 16 presents a more detailed feedback architecture for one UAS in the fleet (denoted *ownship*) in which all elements are certified at design time to the required criticality level. Note in the figure, we continue using the color representation as first presented in Figure 1, where a green block indicates full certification by design time assurance (or a DTA system). Although we explicitly indicate the CLAW, GLAW, FMS and MPS are all DTA systems, the green coloring for all blocks indicate that all other elements are also DTA systems. Again, this is a fundamental assumption. For example, it makes no sense to address the certification level of say, the CLAW, if the airframe itself is not trusted. We will discuss this more when constructing A-G contracts for the physical plant. (Recall, by the legal use of the word *certify*, the blocks in the figure below are not individually certified, rather they are rigorously analyzed/tested through the V&V processes to the point of being trusted or assured to the appropriate criticality level. It is the complete system that is then certified for operational use.)



**Figure 16. Design Time Certified Feedback System for Challenge Problem Ownship UAS**

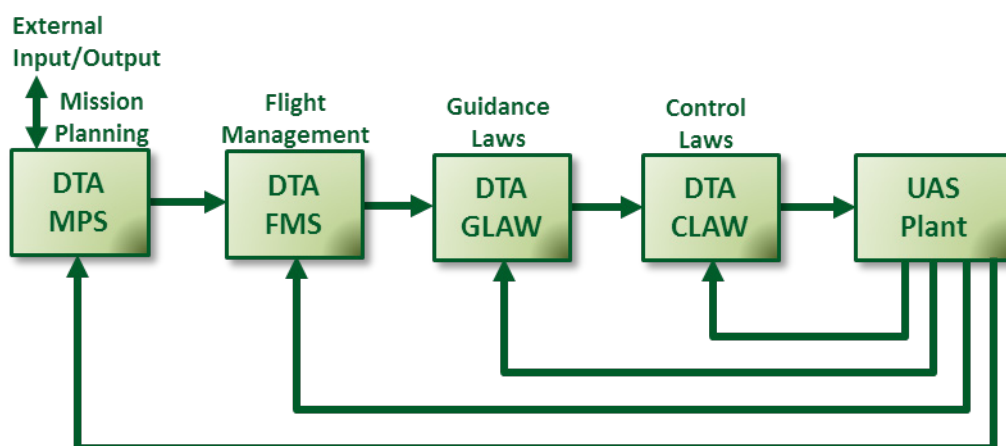
Element by element description of Figure 16:

1. Physical plant: consists of the airframe and engine and all other physical/mechanical devices within the airframe, such as landing gear, payload, etc.
2. Physical actuators: consists of the actuation mechanisms, such as hydraulic systems, and the control surfaces they actuate. For UASs, this will typically include elevators, rudder, ailerons, flaps, speedbrake, and throttle for engine control.
3. Physical sensors: consists of the typical sensor packages for UASs, including GPS, an inertial measurement unit (IMU), compass or magnetometer, radar, altimeter (radar and barometric),

pitot tubes, etc. These sensors are used by the inertial navigation system (INS) to provide the NAV solution, or current state vector for feedback to the different control functions (CLAW, GLAW, etc.). It is assumed that any filtering or smoothing algorithms required are included in the sensor block. The sensor package also includes any other sensors needed to accomplish the UAS mission, which may include electro-optical (EO) or infrared (IR) cameras, or other special devices for ISR missions, for example.

4. Integrated vehicle health and contingency management/redundancy management: as previously discussed, this consists of FDI algorithms to detect and isolate any physical damage or faulted control effectors or sensors (state awareness and real time response). Redundancy management is used to recover from any detected faults. Whether this block is present for UAS platforms is application specific. Note that we are not stating that hardware health management functions are required for RTA systems. If an advanced adaptive hardware fault tolerant system is used in conjunction with RTA protection, then some type of hardware fault detection function will be required. However, for other advanced systems, such a system may not be needed for integration of runtime protection. Since this has not been a focus in this project, we will not continue to present it in subsequent figures. However, IVHM/RM systems may be present in the onboard processing systems.
5. External input/output: all communications into and out of the ownship UAS are performed at the MPS level. This will include intra-fleet communications with fleetmate vehicles as well as with any ground control stations involved in mission coordination. In subsequent figures we will represent these blocks as a two-way arrow attached to the MPS block.
6. CLAW/GLAW/FMS/MPS: our main focus will be on these blocks and their nested feedback interconnections.

The simplified version of the block diagram shown in Figure 16 is shown in Figure 17, which is essentially the same as shown in Figure 5. Here, we 1) display the control effectors block, the airframe and engine block and the sensors block all into one block denoted as the UAS plant; 2) left out the IVHM/RM block for simplicity; and 3) represent the communications block and the fleet and base block as external input/output to/from the mission planning block.



**Figure 17. Simplified Version of Feedback System for Challenge Problem Ownship**



#### 4.4 Expected Approach for Introducing RTA Protected Systems

As the industry begins to introduce advanced systems with runtime protection, such systems will most likely be integrated with existing DTA systems in a careful, build-up manner through single feedback design, experimentation and testing. That is, the first actual real-world design, development and implementation of an advanced inner-loop CLAW will be accomplished with all other elements being DTA systems. Only after the development teams are satisfied that that design is working properly, will they then introduce the next layer of an advanced system, namely the GLAW level, followed by more careful evaluation, experimentation and testing.<sup>2</sup> This build-up approach is depicted in Figure 18, where the first block diagram shows only a RTA protected CLAW is present. Each subsequent block diagram shows the addition of another RTA protected block for the next upstream feedback level. The notation in the figure is, for example:

1. ACLAW = Advanced CLAW
2. RCLAW = Reversionary CLAW
3. RTA\_CLAW M&S = RTA monitor and switch mechanism for the CLAW block.

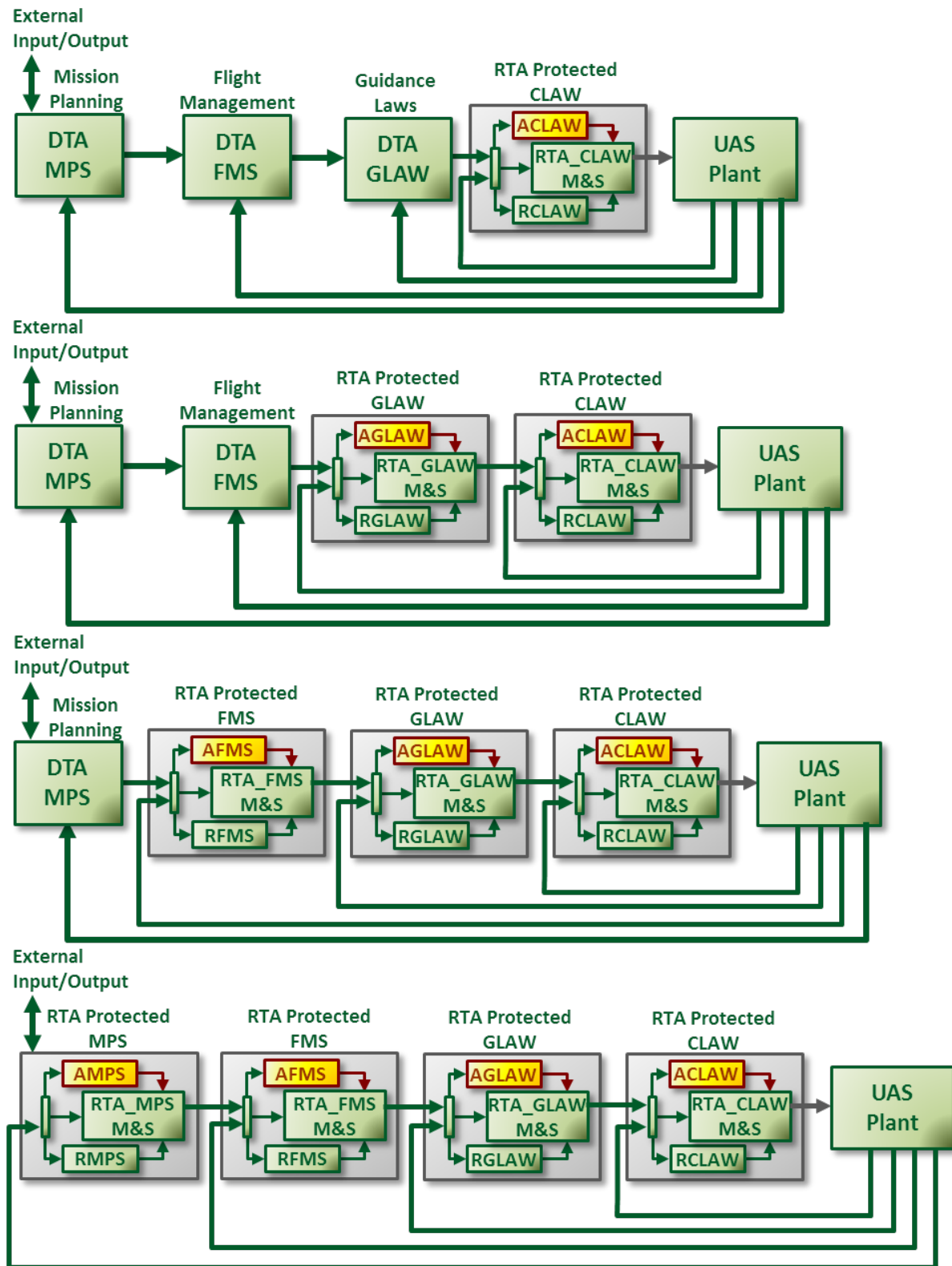
Likewise notation is seen for the other RTA protected blocks.

This build-up approach may proceed as depicted in Figure 18 for some development efforts, but other programs may only focus on one or two particular feedback levels for advanced systems. In our past CPD project, we only addressed an RTA design for an outer-loop autoland guidance system. All other levels, including the inner-loop controller were DTA systems. Figure 19 presents example combinations of interacting DTA and RTA protected systems that may be encountered in any future development or experimental process. The first block diagram only includes an RTA protected GLAW, all other blocks being DTA systems. Again, this was the case for our CPD project. The second block diagram only includes an RTA protected FMS, all other blocks being DTA systems. The third block diagram depicts a development case that focuses only on higher level RTA protected systems, namely the MPS and FMS, with the lower level GLAW and CLAW blocks being DTA systems.

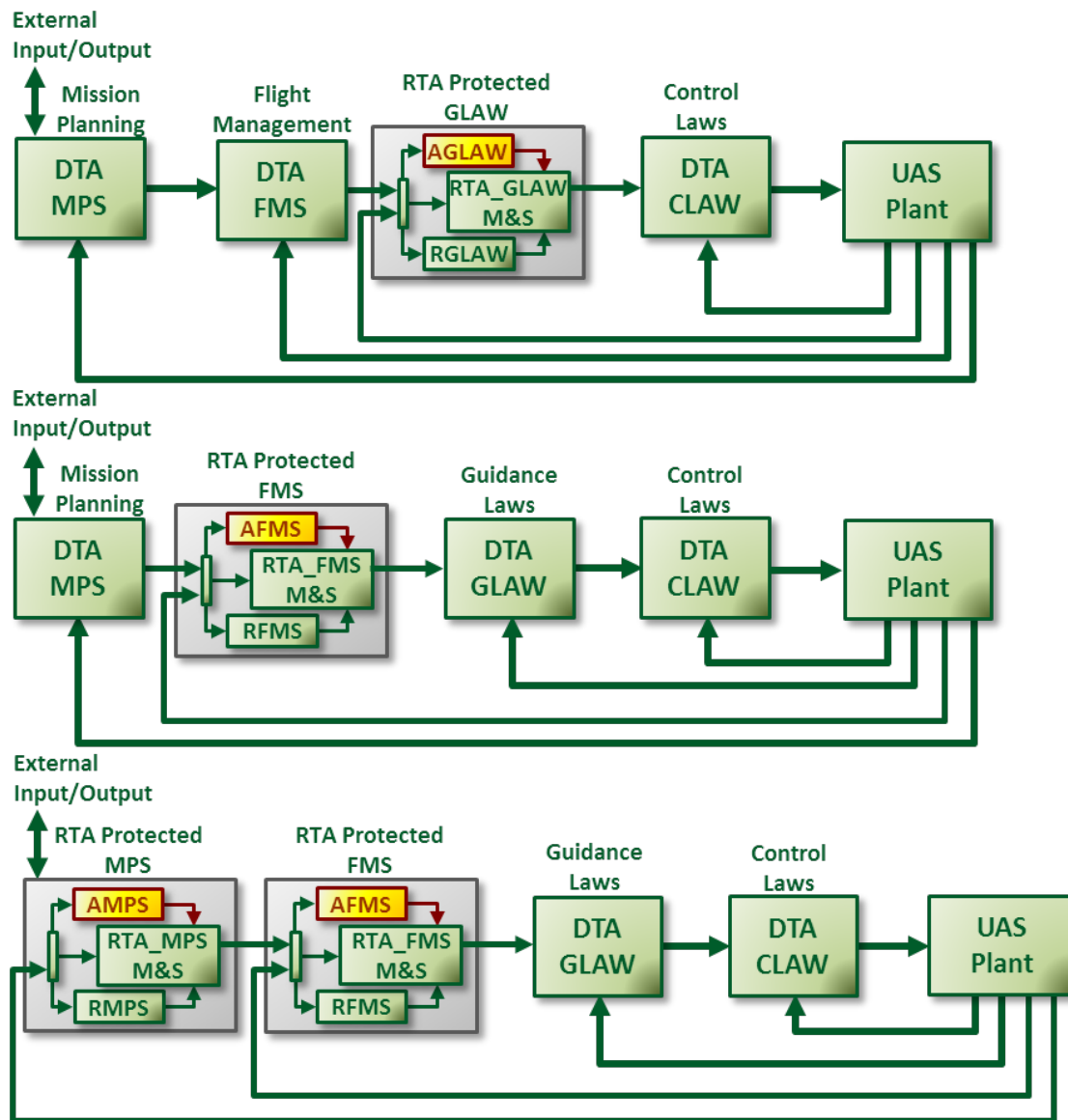
Another reason for mixed DTA and RTA protected systems is that different organizations or design teams will typically develop each different feedback level. It is often the case one company will develop the controller and another company will develop the guidance system, etc. Research organization may purchase an off-the-shelf remote controlled (RC) vehicle that comes equipped with its own inner-loop controller and outer-loop autopilot system which together work with a supplied ground station. The organization may then use the vehicle to flight test some type of experimental flight management system or other software such as a collision avoidance algorithm. In such a case, the advanced experimental blocks are at the higher levels, with DTA systems at the CLAW and GLAW levels.

---

<sup>2</sup> Note, this paradigm is simply conjecture. First implementations of advanced systems with RTA protection may involve multiple feedback levels, depending on the application.



**Figure 18. Build-Up Approach to Introducing RTA Protected Advanced Systems**



**Figure 19. Example Combinations of Interacting DTA and RTA Protected Systems**

Recall that we consider fleets of heterogeneous UAS platforms. That not only implies different vehicle platforms, but also different feedback frameworks, examples of which are depicted in Figure 18 and Figure 19. That is, for example, one vehicle in the fleet may have advanced, runtime protected systems at all feedback levels. It may, however, have to communicate and coordinate with another vehicle that only has an advanced, runtime protected inner-loop CLAW. Or, another vehicle may have only certain guidance or flight management modes that are runtime protected. The implications of having such disparate feedback makeups will be explored when we address the concept of RTA graceful degradation. Not only will we need to explore the implications of mixed DTA and RTA protected frameworks within the single ownership, we will also need to investigate the implications of different mixed frameworks for the interacting vehicles within the fleet. There may be certain scenarios that will require special considerations or special protocols to avoid potentially unsafe operations.

## 4.5 A Modular RTA Protected System Framework

### 4.5.1 Motivation

After investigating a number of aspects of interacting DTA and RTA protected feedback systems, we recommend a modular RTA protected system framework, and that is what we have developed for the challenge problem in this program. As just discussed, research and development programs may have a number of variations or combinations of DTA blocks and RTA protected blocks resulting from a certain development focus. Or, often research projects involve off-the-shelf test vehicles with input/output structure already dictated and with certain elements already proven DTA systems. Further, design and test cycles often involve rapid *swap in/swap out* of experimental advanced elements (e.g., one flight test may involve only an advanced FMS with RTA protection, followed immediately with the addition of an advanced GLAW with RTA protection, etc.). Also, recall, there may be external code that needs to be integrated into the system, such as certified IVHM or collision avoidance systems (which may have their own design cycle testing stages). Therefore, for these types of scenarios we seek systems at each feedback level design to be able to be connected to either a DTA upstream or a DTA downstream block or an RTA protected upstream or an RTA protected downstream block, and other external code. This fundamentally implies a *plug and play*, modular framework. Last, this modular approach naturally supports the compositional reasoning approach to certification. Full certification of each element in a nested feedback architecture (whether DTA or RTA protected) will substantially reduce the burden of full certification of the entire interconnected system.

### 4.5.2 Input/Output Protocol for a Modular RTA Protected Framework

A modular RTA protected system is one in which both the advanced and reversionary systems must accept the same inputs (or input vector, or data packet) and generate output that is equivalent in its definition. That is, the variables in the output vector should be delivered in the same order, have the same definition for each variable, and have equivalent units. Their numerical values may be different, but the definition of the output should be exactly equivalent. In this manner, the commands into the RTA protected system can be from any appropriate upstream block, be it a DTA system or a RTA protected system. Likewise, whether in advanced or reversionary mode, any appropriate downstream block should be able to accept the output of the RTA protected system, be it generated from the advanced or reversionary system. Therefore, the downstream block could be either a DTA or RTA protected system. There would be no need for any special consideration or changes made to the interface between the two levels if the RTA monitor commanded a switch to the reversionary system.

Therefore, what is *not allowed* is a separate, specific input vector for the advanced system, generated by its corresponding upstream advanced system, or a separate, specific output vector generated by the advanced system at the current level to be delivered only to its corresponding downstream advanced system. Likewise, separate, specific input or output vectors should not be generated by corresponding reversionary systems. Such systems result in a much more tightly coupled architecture and do not allow for the *plug and play* modularity we seek. This type of system is depicted in Figure 20. It is illustrated that advanced system No. 2 generates its own output specifically for the downstream advanced system No. 1. Likewise, reversionary system No. 2 generates its own output specifically for the downstream reversionary system No. 1. Note that the output of advanced system No. 2 still needs to be checked by the RTA monitor even if it

Separate pathways for I/O between each advanced and each reversionary systems

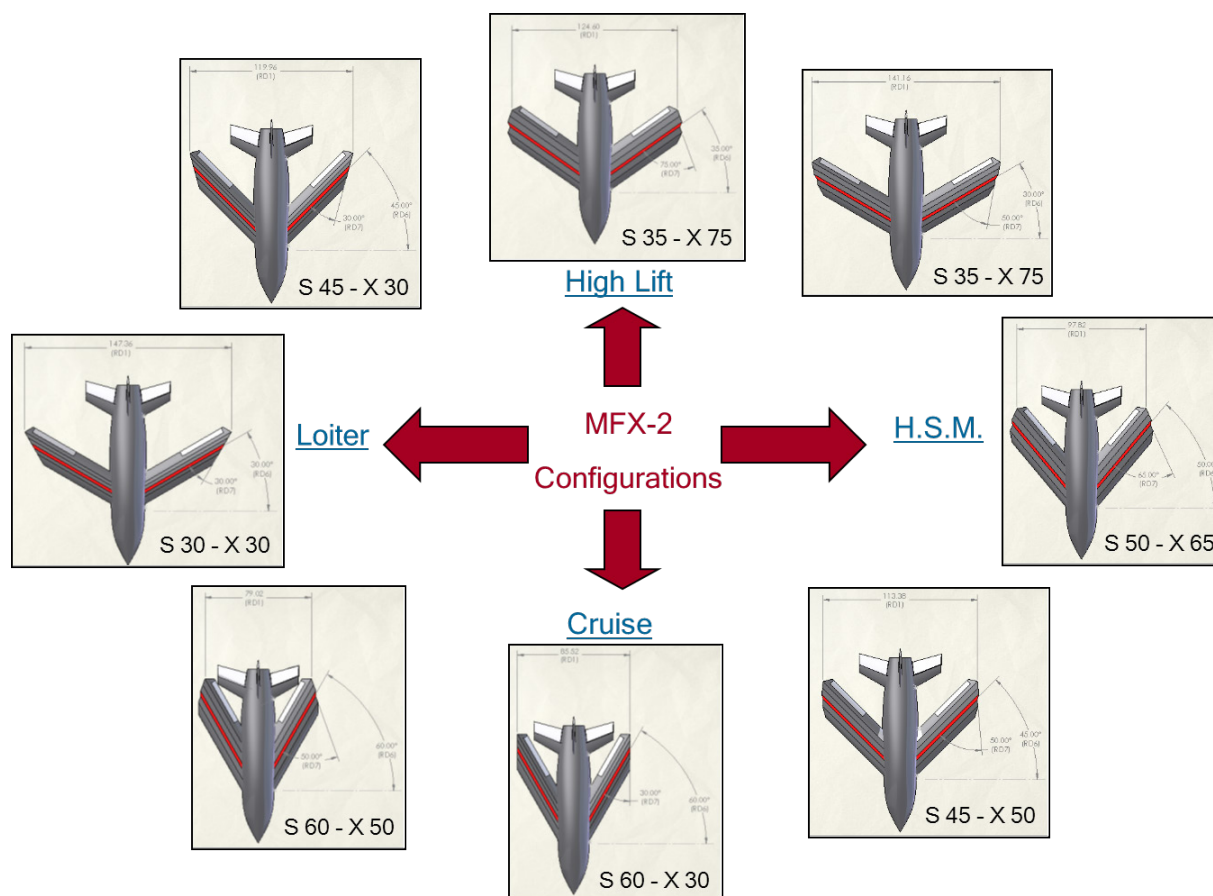
The diagram illustrates a dual-channel architecture for two RTA Protected Systems (System #1 and System #2). Each system contains an Advanced System (yellow box) and a Reversionary System (green box). The Advanced System is connected to the RTA Monitor & Switch (green box) via a red line. The Reversionary System is connected to the RTA Monitor & Switch via a green line. The RTA Monitor & Switch is connected to the Plant or System being Controlled (green box). The diagram shows separate pathways for I/O between each advanced and each reversionary system, ensuring redundancy and safety.

The modular approach does make the design of the elements within the RTA protected block more difficult. For example, additional input or output *conditioning* may be needed. Consider an FMS block that delivers waypoints to the guidance law. Typically, the reversionary guidance law may be an industry standard waypoint follower, so it could directly accept the FMS generated waypoints as inputs. However, common to advanced guidance laws are that they work with continuous trajectory path commands, not waypoints. In this case, the designers of the advanced guidance system will need to first convert the waypoints that are delivered from the FMS block to an equivalent continuous path trajectory, and then have the guidance law work with that derived path. In summary, even though additional steps may need to be taken, the advantages of having modular capabilities in RTA protected systems is that they can easily replace DTA systems without any restructuring of the interfaces or other special considerations. This is quite advantageous especially during design iteration and rapid testing stages.

## 5 RTA Protection Applied to Inner-Loop Control Systems

### 5.1 General Description of Morphing Wing Vehicle

For the inner-loop control challenge problem, a simulation of a morphing wing model was used. This that was first developed in a past AFRL SBIR Phase II program [Gandhi 2008] and was known as the MFX-2 platform. A physical mock-up of the vehicle was constructed for wind tunnel and wing morphing experiments in that program. Through a series of jack-screw mechanisms, the wing on this vehicle was capable of changing its planform in flight to optimize the aerodynamic properties for various flight conditions (e.g., minimize drag for cruise or high speed maneuver, maximize lift for high lift and loiter, etc.). Figure 21 presents a summary of the general baseline planform configurations for this vehicle. The wings can continually morph from one baseline shape to another while in flight. The planform shape is defined by two angles, a wing sweep angle, denoted in the figure as 'S' and an internal morphing mechanism angle, denoted as chi, or 'X' in the figure. The angle chi can change the wing chord length, hence the planform shape of the wing.



**Figure 21. Morphing Wing Baseline Planform Configurations**

Barron Associates developed an advanced, adaptive inner-loop controller for this vehicle that was capable of real-time identification of its aerodynamic properties as the wing morphed from one configuration to another. At each controller update, the identified model parameters were used to solve for updated feedback gains using a model predictive control scheme known as

receding horizon optimal (RHO) control. In this manner, the advanced controller eliminated the need to derive thousands of linearized set point controllers that would otherwise be needed in a classical gain-scheduled approach.

The 6-DOF governing equations of motion for this vehicle, its force and moment model, and numerical values for all model parameters at one example flight condition (straight and level flight at loiter planform configuration) are given in Appendix C. The inputs to the inner-loop controller are commanded altitude, bank angle and airspeed. The outputs of the controller are control surface deflection commands and thrust command to the engine.

## **5.2 Reversionary CLAW Description**

For the reversionary control law design, an obvious choice for a baseline controller is a classical control design about an equilibrium point specifically for one of the planform configurations shown in Figure 21. Although our studies focused only on a baseline controller designed for the loiter configuration, notionally, a number of baseline controllers could be constructed for several of the planform configurations shown in this figure. A key reason that this particular model was chosen for investigating RTA applications at the inner-loop level was that this problem gives a clear example of an advanced controller that has a larger operating region than the baseline controller, requiring inclusion of one or more transition control systems. The advanced controller can operate at any condition within its valid flight envelope, under any planform shape, whether it is at one of the baseline planforms in Figure 21, or at an intermediate shape while transitioning from one baseline to another. However, the baseline controllers within the reversionary system can only operate around one of the baseline planform configurations. If the RTA monitor determines that the advanced controller needs to be shut down while in a state other than one of the baseline planform configurations, then one or more transition controllers will need to drive the state to an operation region around a chosen baseline planform shape.

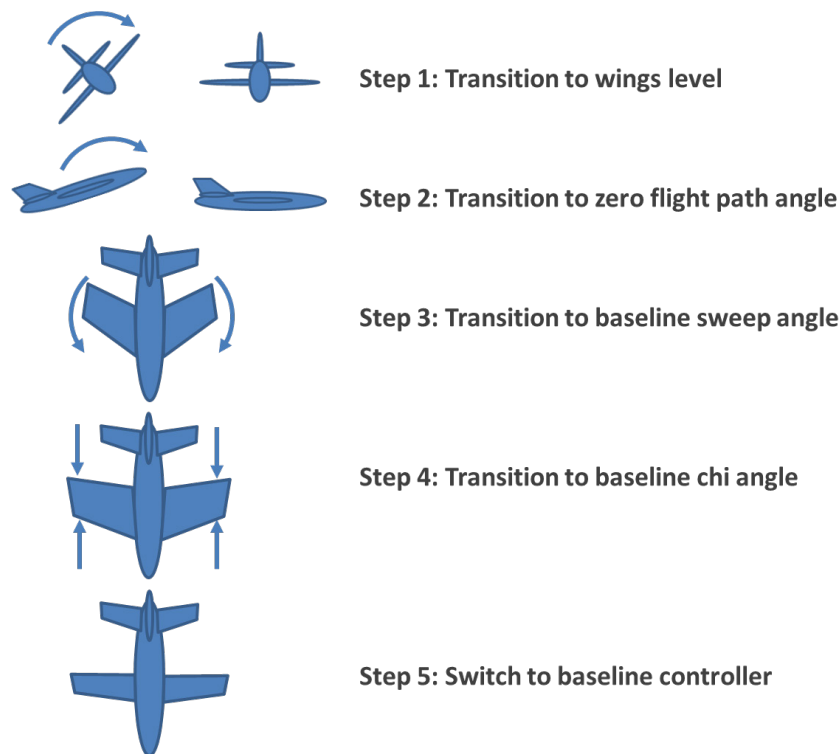
### **5.2.1 Transitions Controller Designs**

The approach we took was to design transition controllers using classical, gain scheduled linear methods which can be certified at design time. However, for the transition controllers to be able to cover the complete operating envelope of the advanced control system, many thousands of set point designs would have to be constructed, which would clearly negate the advantages offered by the advanced, adaptive controller. To address this curse of dimensionality, we formed a transition approach that considerably reduces the state space required to be covered by the transition controllers. To do this, once the advanced controller is deactivated, any commands from the upstream guidance system are ignored during the transition procedure, and specific reversionary commands are followed until the operating state reaches a designated baseline region. Once the baseline region is reached, then guidance system command-following is reestablished to drive the baseline controller.

The transition procedure is depicted in Figure 22, which shows five main steps:

Step 1: The first transition controller (or series of gain scheduled controllers) must take the state of the system from any point where the advanced controller was deactivated to a state with zero bank angle (or wings-level flight condition). Although this first step must cover the entire operating region of the advanced controller, these transition

controllers have only one single purpose – to drive the bank angle to zero. The assumption here is that this single design requirement should help to expedite the construction of this set of gain scheduled controllers.



**Figure 22. Transition Procedure from Advanced to Baseline Control**

- Step 2: The next set of transition controllers are then used to drive the flight path angle to zero (or constant altitude flight condition). Here, however, the number of required set point designs is considerably reduced since only wings-level flight need be addressed. Here too, the single design requirement of driving the flight path angle to zero should help to expedite the construction of the controllers.
- Step 3: The next set of transition controllers are then used to maintain stable flight as the sweep angle is transitioned to the chosen baseline sweep angle. The number of required set point designs is further reduced since only wings-level/constant altitude flight need be addressed.
- Step 4: The last transition step is to maintain stable flight as the chi angle is transitioned to its baseline value. The number of required set point designs is even further reduced since only wings-level/constant altitude flight/baseline sweep configuration need be addressed.
- Step 5: The final step is to activate the baseline controller that operates at the baseline wing planform configuration. Notionally, the baseline controller then has the capability to fly the vehicle to a safe steady-state condition, or to fly to its home airbase to safely recover the vehicle.



Appendix C also presents the details of the transition controller design procedure that was adopted from [Seto 1999]. These transition controllers were used in the numerical experiments, demonstrating the benefits of the RTA system. These results are presented at the end of this chapter.

### 5.3 Type Safety at Inner-Loop Control Law Level

Recall that we define the set  $S_{Dsafe}$  as the set of states in which it has been determined (through analysis, simulation studies, etc.) that the plant or system can operate safely and correctly. At the inner-loop level, the plant is the physical aircraft system, including the airframe, engine, actuators and sensors. Therefore, states within  $S_{Dsafe}$  are those in which the structural and aerodynamic properties of the vehicle are such that it can operate continually at this point without:

- Aerodynamic instability (flow separation)
- Attitude upset condition
- Mechanical failure (e.g., structural overloading)
- Actuator rate, deflection or power exceedances

In general, any condition that can cause the vehicle to be unable to fly correctly or safely or that can cause physical damage to the structure of the vehicle are states outside of  $S_{Dsafe}$ . Again,  $S_{Dsafe}$  will be a multi-dimensional region in state space because of many interdependencies within the critical states of the aircraft. For example, stall angle-of-attack will be a function of flap deflections, stall airspeed a function of vehicle weight/payload configuration, maximum load factor a function of roll rate, etc. Further, parameters such as Mach number, altitude, atmospheric properties, etc., will also influence the maximum allowable values for many safety-critical parameters. With this, we can now formally define safety levels that are used to determine the switching condition protocol for an RTA protected inner-loop system. We will use the capital letter ‘C’ to denote that these safety definitions are for the control loop level.

**Definition 5.** Inner-Loop Control Type Safety - A point  $x_0$  in the state space  $S$  is:

- **CType I Safe** if that point lies inside  $S_{Dsafe}$ .
- **CType II with set  $Q_c$  and time  $T_c$  Safe** if all of the following hold:
  - d) Point  $x_0$  is CType I safe,
  - e) Upon switching to the reversionary **CLAW**, the state trajectory can converge to at least one point in a *desired* set  $Q_c$  within a given *desired* time  $T_c > 0$ ,
  - f) The state trajectory from the point of switching to the reversionary system to the point of reaching the set  $Q_c$  is entirely contained within the CType I safe region.
- **CType III with Period  $\tau_c$  Safe** if all of the following hold:
  - c) Point  $x_0$  is CType II safe,
  - d) Every possible output of the advanced system for a time period  $\tau_c$  results in a state trajectory entirely contained within the CType II safe region.

For our morphing wing example, let us define the desired region  $\mathcal{Q}_c$  to be achieved at the point in which the transition controllers have achieved wings-level, constant altitude flight. (Note, for other designs,  $\mathcal{Q}_c$  could be defined differently, such as the point at which the state has reached the operating region of the baseline controller. Again, how this region is defined is specific to the reversionary controller design.)

Since the inner-loop dynamics operate at a very high bandwidth, the period  $\tau_c$  will be relatively small in value (of the order of 0.01 seconds, for example). Therefore, the RTA monitor should be updated at least as fast as the inner-loop controller update.

## 5.4 A-G Contracts at Inner-Loop Control Law Level

Again, to build a safety case argument, we turned to using the compositional reasoning approach and constructed A-G contracts for each element in the feedback loops. Once we have analyzed and constructed the A-G contracts at the inner-loop level, we can view the inner-loop in a composite manner when we next construct A-G contracts at the guidance law level, and so on for each successively higher level feedback loop.

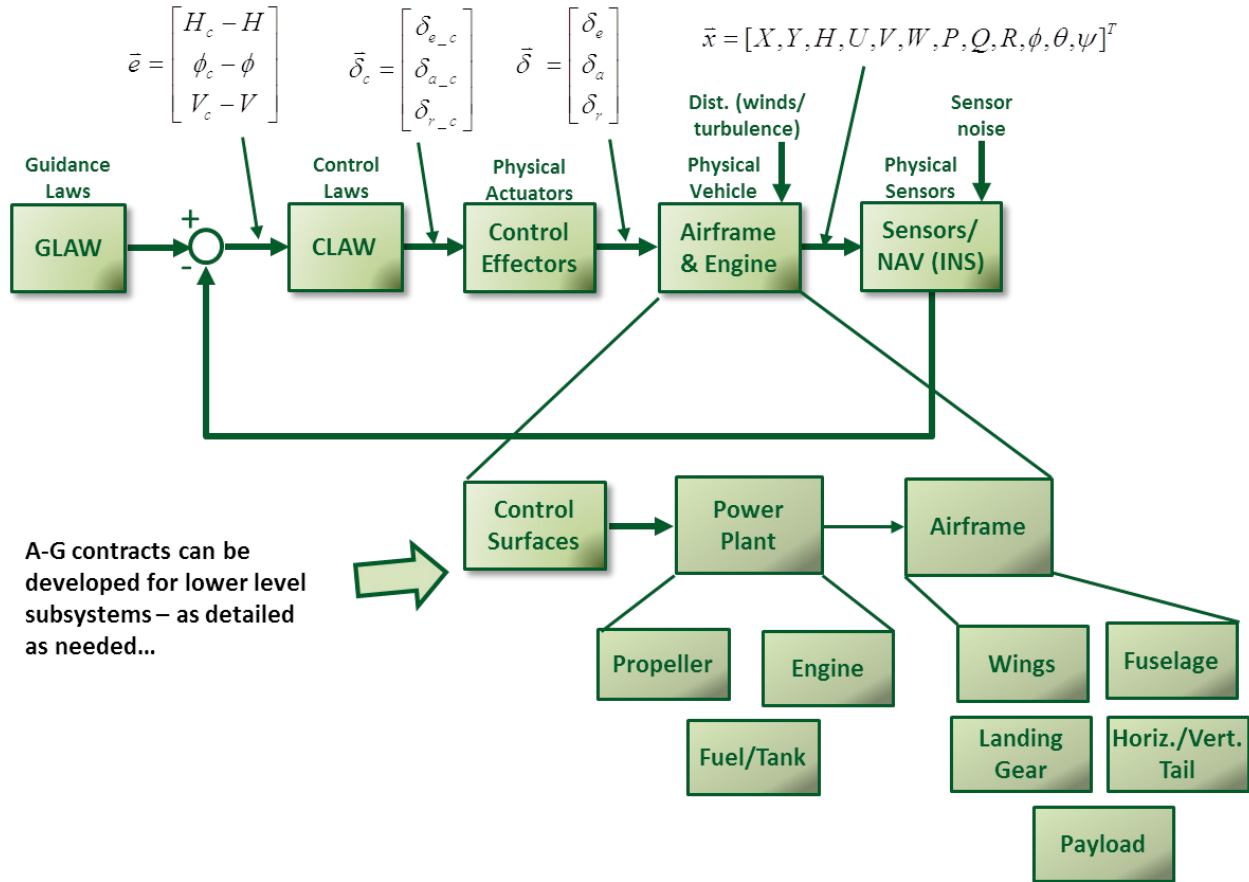
Figure 23 shows the inner-loop feedback level, labeling the input/output relationships for each block in the feedback loop. Note that in a full-up detailed design/development program, A-G contracts would be developed for all lower level subsystems within the overall aircraft platform, involving, for example, the actuators and their respective control surfaces, or the various sub-elements of the power plant (propeller, engine, fuel/tank, etc.), or the airframe, (wings/tail, fuselage, landing gear, payloads, etc.). We will not, however, develop such detailed A-G contracts here, but instead show how the overall process would be accomplished, focusing on the main elements in the feedback loop.

### 5.4.1 Assumptions on Inputs from Upstream Guidance System

We start with constructing the assumptions on the inputs from the upstream guidance system, or the GLAW commands into the inner-loop. Again, these commands consist of an altitude command, a bank angle command, and an airspeed command. Therefore, there will be assumptions placed on these inputs that they will not violate minimum and maximum values, and that their rates of change will not violate minimum and maximum values. These assumptions are listed as:

$$\begin{aligned}
 0_{(AGL)} &< H_c < H_{\max} \\
 |\phi_c| &< |\phi_c|_{\max} \\
 V_{\text{stall}} &< V_c < V_{\max} \\
 \text{if } \dot{H}_c < 0, \dot{H}_{sr\_min} &< \dot{H}_c \\
 \text{if } \dot{H}_c > 0, \dot{H}_{clmb} &< \dot{H}_{clmb\_rate\_max} \\
 |\dot{\phi}_c| &< \text{max roll rate} \\
 |\dot{V}_c| &< |\dot{V}_c|_{\max}
 \end{aligned} \tag{1}$$

There may also be other assumptions on, for example, frequency content. Dynamic systems have a natural bandwidth, and if commands to that system are at a frequency greater than the bandwidth of the system, then there can be no guarantee that the system can accurately follow the commands (see Figure 14). Runtime methods exist that can estimate frequency content and such algorithms could be used online to check that such frequency requirements are being met.



**Figure 23. Development of A-G Contracts at the Inner-Loop Level**

#### 5.4.2 A-G Contracts for Control Effectors

The first assumption here is that the actuators and control surfaces are working properly and that there is no damage or faulted hardware. For the morphing wing model, the control effectors consist of elevators, ailerons and a rudder. Here too, there will be assumptions placed on the inputs (commands from the control law) that they will not violate minimum and maximum values, and that their rates of change will not violate minimum and maximum values. These assumptions are listed as:

$$\begin{aligned} \delta_{e\_min} \leq \delta_{e\_c} \leq \delta_{e\_max}, \quad \delta_{a\_min} \leq \delta_{a\_c} \leq \delta_{a\_max}, \quad \delta_{r\_min} \leq \delta_{r\_c} \leq \delta_{r\_max} \\ \dot{\delta}_{e\_min} \leq \dot{\delta}_{e\_c} \leq \dot{\delta}_{e\_max}, \quad \dot{\delta}_{a\_min} \leq \dot{\delta}_{a\_c} \leq \dot{\delta}_{a\_max}, \quad \dot{\delta}_{r\_min} \leq \dot{\delta}_{r\_c} \leq \dot{\delta}_{r\_max} \end{aligned} \quad (2)$$

Also, assumptions may be placed on frequency content, as there will be physical limitations on the bandwidths of the actuation systems. Therefore, the frequency content of the commanded

inputs to the control effectors should not be greater than the bandwidth (or break frequencies) of the actuation systems.

As long as the assumptions hold for the control effectors, then the guarantees, in general, are that they should be able to accurately follow their commands. There may be specific performance requirements that must hold, such as on maximum settling time:

$$T_{settle} \leq T_{settle\_max} \quad \forall t \quad (3)$$

which is a measure of the *responsiveness* of the actuation system. Another guarantee may be that any overshoot from a commanded value should not exceed a certain value, or that maximum error between the command and response never exceed a certain value, as:

$$|\delta_c - \delta| \leq \varepsilon \quad \forall t \quad (4)$$

Last, there may also be a guarantee placed on the dynamic characteristics of the control effectors such that they follow, for example, a second order system to within certain defined tolerances, as:

$$\ddot{x}(t) + 2\zeta\omega\dot{x}(t) + \omega^2 x(t) = 0, \quad \zeta = \zeta^*, \quad \omega = \omega^* \quad (5)$$

#### 5.4.3 A-G Contracts for Sensors

The assumptions on the sensor systems are that their hardware is all working properly and has been properly calibrated and maintained. Further, the sensors are only being used in their intended environments. For example, an electro-optical camera cannot be used at night or in low lighting conditions, or in poor visibility conditions, such as in dust, smoke, or clouds. Another example is that a radar altimeter is only accurate at low altitudes, typically below 2,000 ft AGL. Altitudes above 2,000 ft should be determined using a barometer. There may also be assumptions placed on noise that can affect the sensor output, such as its density function characteristics or magnitudes.

As long as the assumptions hold for the sensors, then the guarantees, in general, are that they should be able to accurately measure their sensed inputs. This may be measured through statistical means, such as maximum values for filter residuals, or maximum values on peak errors, etc. These guarantees would be obtained from the sensor vendors, for example.

#### 5.4.4 A-G Contracts for the Airframe/Engine

The assumptions on the physical aircraft (airframe and engine) are that all equipment is working properly and that it is only being flown in its intended environments, which will typically place limits on winds aloft and average levels of turbulence, for example. Other assumption would be that aerodynamic limits are not violated, attitude stability is maintained, no excessive loss of lift, no over-speed conditions are encountered, structural loading limits are not being violated, actuator hinge moment limits are not violated, etc. These may be expressed as:

$$\begin{aligned}
&\text{Aerodynamic: } \alpha < \alpha_{\text{stall}} = f(V, \rho, \dots), \quad \beta < \beta_{\text{max}} = f(V, \dots) \\
&\text{Attitude stability: } [P, Q, R] \leq [P, Q, R]_{\text{max}}, \quad |\phi| \leq |\phi|_{\text{max}}, \quad \theta_{\text{min}} \leq \theta \leq \theta_{\text{max}} \\
&\text{Structural: } N_z < N_{z_{\text{max}}}, \quad V \leq V_{\text{max}}, \quad \bar{q} < \bar{q}_{\text{max}}
\end{aligned} \tag{6}$$

Note that the combined assumptions on the physical plant (airframe, engine, actuators and sensors) are equivalent to the assumption that the aircraft is flying within the  $S_{\text{Dsafe}}$  region of the state space. Therefore, as long as all the stated assumptions on the physical plant hold, then the state is guaranteed to lie within the CType I safe space.

As long as the assumptions hold for the airframe/engine, then the main guarantee is that the aircraft dynamics will be as expected (as modeled) to within some uncertainty bound, or:

$$\dot{x} = f(x, \delta, \text{Dist}) + \Delta, \quad \text{or} \quad \dot{x} = Ax + B\delta + \text{Dist} + \Delta \tag{7}$$

#### 5.4.5 A-G Contracts for the Inner-Loop Controller

We will summarize the A-G contracts that must hold for the inner-loop controller, be it an advanced, reversionary, or DTA controller (that is, in any of these cases, the A-G contracts must hold, although specific values or measures for, for example, performance requirements may be different for different controllers). Since the controller is designed with the fundamental assumptions made on the rest of the elements shown in Figure 23, the assumptions here are that all A-G contracts for all the other elements around the feedback loop hold. That is,

- i. All equipment working properly
- ii. Environment assumptions hold
- iii. Aircraft dynamics are as expected to within some uncertainty delta
- iv. States/parameters lie within expected bounds
- v. Actuator rate/deflection limits hold (no saturation, no integrator wind up)
- vi. Guarantees on sensors and control effectors hold
- vii. Assumptions on guidance commands hold.

As long as these assumptions hold, then the guarantees are that the controller:

- i. Maintains attitude stability of the airframe for all time,
- ii. Achieves the required tracking performance for all time,
- iii. Stability and performance robustness is maintained for all time, and
- iv. Stated assumptions on command inputs to the control effectors hold for all time.

For tracking performance, this may be ascertained by certain defined characteristics of the error vector, such as that a norm bound (e.g., peak value) holds, or a settling time is achieved, etc. (again, the measures will depend on the specific control law design approach). This maybe expressed as:

$$\begin{aligned} \bar{\delta}_c = \begin{bmatrix} \delta_{e-c} \\ \delta_{a-c} \\ \delta_{r-c} \end{bmatrix} &= f^*(f(x, Dist, \Delta)), \text{ s.t. } |\bar{e}| < \varepsilon \quad \forall t, \quad \bar{e} = \begin{bmatrix} H_c - H \\ \phi_c - \phi \\ V_c - V \end{bmatrix} \\ \text{or, } \|\bar{e}\|_n &< \varepsilon \end{aligned} \quad (8)$$

where the function  $f^*$  defines the control law and  $\bar{e}$  is the feedback error vector.

Stability and performance robustness guarantees will be with respect to specified measures of plant modeling uncertainties, plant disturbances (e.g., turbulence magnitude) and sensor noise characteristics.

Last, stability, performance and robustness are all guaranteed while delivering commands to the control effectors that do not exceed rate or deflection limits and have valid frequency content.

#### 5.4.6 A-G Contracts for the CLAW RTA System

For the reversionary controller, the A-G contracts will be equivalent in structure to the inner-loop controller in general. However, specifically, the reversionary controller also guarantees that at least CType I Safety holds for all time, and that the desired region  $\mathbf{Q}_c$  is achievable within time  $T_c$  when it is first activated by the RTA monitor and switch mechanism.

Also, as previously discussed, the reversionary controller's guaranteed performance characteristics may be less, by some measure, than the advanced controller. If this is the case, then if the RTA monitor commands a switch to the reversionary controller, this will change the controller's A-G contract at the inner-loop level. This change or reduction in the inner-loop controller's performance capabilities must be communicated to all upstream feedback levels, as this can affect how well the aircraft can follow a commanded path or whether it can reach a certain mission objective within a required time period, for example.

For the RTA monitor and switch mechanism block, the main assumptions are that the information input to this block is valid (correct to within some accuracy tolerance) and that it at least starts its operation while the system is within the CType III safety envelope. That is, the RTA system cannot make any guarantees of safety if it does not start monitoring the system state until after it has crossed the CType III safety boundary. This is not a limiting assumption since nominally the RTA system will be activated at the time the UAS vehicle is launched at mission start and presumably it would not be launched in unsafe conditions.

As long as the above assumptions hold, the guarantees for the RTA monitor and switch mechanism are that:

- i. Loss of CType III safety is always detected at first occurrence, while the system is in the CType II safe region, and
- ii. The reversionary controller is activated when loss of CType III safety is detected.

As a final note on A-G contracts, there can be no trusted A-G contracts for the advanced controller (at least not at the required criticality level of the RTA protected block). That is, there is no guarantee of safety or performance when the advanced controller is active. For this reason,

the RTA monitor must continually check for ensuing loss of safety or required minimum performance while under active advanced control. In fact, the RTA monitor can be thought of as a runtime A-G contract checker. The checks the RTA system must perform at the inner-loop level are listed in the next subsection.

## **5.5 Inner-Loop RTA Checks and Switching Conditions**

Recall, the general checks the RTA monitor must perform were listed in Subsection 3.4.1. We can now specify these checks for the inner-loop feedback level.

1. Safety check: At each update to the inner-loop RTA monitor, it checks the system state to determine if it has left the CType III safe region. If so, the advanced controller is shut down and control is switched to the reversionary controller.
2. Output/environment check: The inner-loop RTA monitor checks that the actuator commands generated by the advanced controller do not violate any constraints imposed on the control effectors, and in turn, on the airframe/engine plant. Again, this will involve actuator rate and deflection limits and frequency content of the commanded deflections.
3. Performance check: The RTA monitor checks that the advanced controller is achieving its minimum required command tracking performance on altitude, bank angle and airspeed. Again, these checks may include transient response specifications, such as rise time for commanded step inputs, or settling times for disturbance rejection measures. Steady state norm bounds on tracking errors are another example of a common performance measure.
4. Input/environment check: The RTA monitor should also check that inputs or commands to the inner-loop feedback level from the guidance system do not violate any constraints imposed on the inputs. Such constraints will include maximum/minimum values, rate limits, frequency content, and possibly other statistical measures on the altitude, bank angle and airspeed commands. If sensor information can be used to determine the operating environment, then this additional check should be performed to ensure that the aircraft is not be driven to an environmental state that it was not designed to operate in.
5. System hardware health check: recall that many of the assumptions called for hardware to be working properly or for information to be valid and correct. These assumptions address hardware and information integrity. An IVHM system may work in conjunction with an RTA system. Here, the RTA monitor will continually check with the IVHM system to determine if a control effector or sensor has failed, or if information passed within the onboard processors has been compromised by, for example, a security breach. If this is the case, then mitigation strategies will need to be in place to correct the anomalies in some manner and this will be communicated to the RTA system. Such mitigation strategies may be to 1) allow the adaptation capabilities of the advance controller to continue, or 2) activate a redundancy management system to recover from a faulty sensor, or 3) have some cyber security measures in place to discover and mitigate any malicious attack on the sensor pathways, for example.

## **5.6 Inner-Loop Experimental Results**

The focus of the inner-loop experiments was on the RTA system checking for loss of CType III safety. Constructing full-up, detailed CType I to III safety boundaries was beyond the scope of

this project and would require substantial analysis and simulation studies. Rather, preliminary simulation studies were conducted to approximate CType I to III safety boundary values for a select set of parameters at one flight condition. This targeted set of experiments was performed to showcase the benefits of RTA monitoring.

To simplify the analysis, the longitudinal and lateral dynamics were investigated separately and initial simulations were performed to approximate the margins between the CType I to III boundaries. From background material on the MFX-2 model, we defined CType I values for a number of select critical parameters. A number of simulations were then performed at a particular initial condition, switching from the advanced controller to the reversionary controller. For each simulation run, the value of one of the critical parameters was incrementally increased until it was observed that its value went beyond its defined CType I value (due to momentum of the vehicle and/or control mode switching transients). From these results, we approximated the parameter value at which the RTA monitor should switch to the reversionary controller to ensure that the parameter would not exceed its CType I value. We recognize that this was a simplified procedure and we did not vary more than one parameter at a time, ignoring the multi-dimensional nature of the switching condition boundary. That is, the value at which the RTA should activate the control mode switch will be a function of several other states, such as airspeed, wing loading, bank angle, etc. We did not construct this more complex, multi-variable switching function. Further, we did not take into account extra margin needed for variations in environmental conditions or modeling uncertainty. Crude as they were, these values were determined simply for demonstration purposes.

Table 4 shows the CType I to III values for angle-of-attack, normal acceleration and maximum airspeed. For angle-of-attack and normal acceleration we determined both their respective maximum and minimum values. For airspeed, we only focused on the vehicle's overspeed condition. There will also be a minimum set of airspeed values that will result in stall conditions; however, we did not focus on that case. Other switching condition values were also determined (or approximated), such as maximum control surface deflection angles.

**Table 4. Longitudinal Case Study: CType I, II and III Values**

| <b>Parameter</b>    | <b>CType III Value</b> | <b>CType II Value</b> | <b>CType I Value</b> |
|---------------------|------------------------|-----------------------|----------------------|
| Angle-of-Attack     | -2.0 - 14.6 deg        | -2.6 – 14.7 deg       | -3.0 – 15.0 deg      |
| Normal Acceleration | -1.75 – 3.75 g         | -1.8 – 3.8 g          | -2.0 – 4.0 g         |
| Maximum Airspeed    | 226 kts                | 227 kts               | 230 kts              |

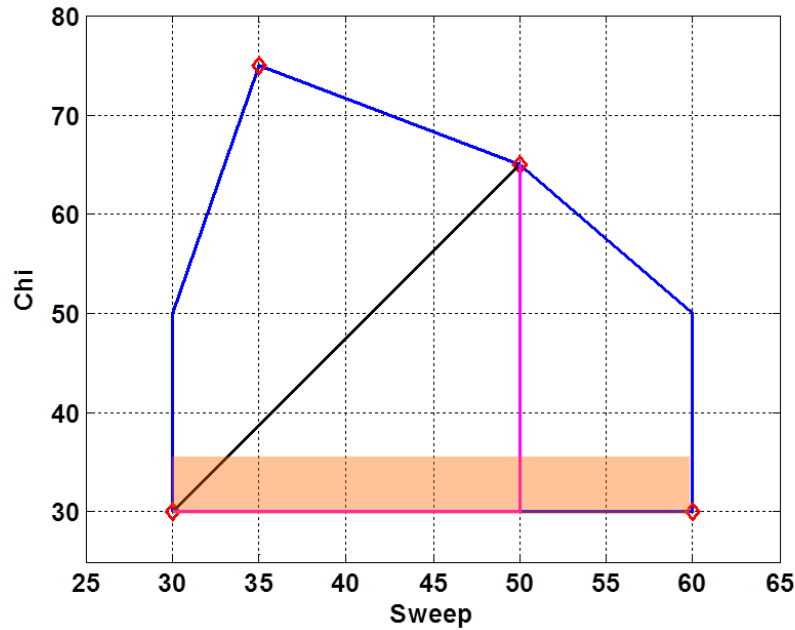
Similar studies were also performed for the lateral dynamics. The amount of bank angle overshoot was investigated for a number of control mode switching values. From these studies, it was determined that there is significant coupling between defined maximum bank angles and required aileron deflection angles to arrest the aircraft's roll. This highlighted the multi-dimensional nature of the switching condition boundary.

### **5.6.1 RTA Demonstration: Mismanaged Wing Morphing**

In this demonstration, a wing morphing procedure was erroneously conducted during an aggressive maneuver resulting in activation of the reversionary controller. This case mimics an error in the advanced logic that manages the morphing of the wing (we presume here that such



advanced logic exists, although we did not actually encode such a system for these studies). The simulation was initialized in the Loiter configuration ( $S = 30, \chi = 30$ ). At 5 seconds, the advanced controller begins a transition to the High Speed Maneuver configuration ( $S = 50, \chi = 65$ ). Instead of following the shortest straight line path (the black path in Figure 24), an error results in a morph that first changes sweep and then changes chi (the magenta path in Figure 24). As the morph is initiated, a climbing high bank turn is also commanded. The shaded region in Figure 24 is not compatible with a climbing high bank maneuver. As a result, at approximately seven seconds, proximity to elevator saturation triggers the RTA switch and the transition controller is enabled at approximately 7.5 seconds.

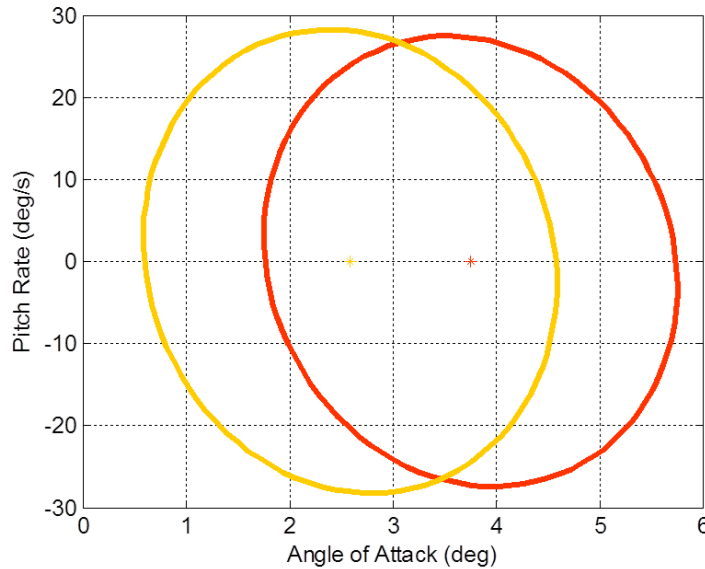


**Figure 24. RTA Demonstration: Mismanaged Wing Morphing Procedure**

The transition controller follows the process outlined in Subsection 5.2.1:

1. Stage 1: Go to wings level at the current wing configuration ( $S = 41, \chi = 30$ ).
2. Stage 2: Zero flight path at the current wing configuration ( $S = 41, \chi = 30$ ).
3. Stage 3: Transition to baseline chi while maintain the current sweep ( $S = 41, \chi = 30$ ) (the wing is already at the baseline value).
4. Stage 4: Transition to baseline sweep while maintaining the baseline chi ( $S = 30, \chi = 30$ ).

Control is passed to the baseline controller when the wing is back in the Loiter configuration. The linear feedback laws that comprise the transition controller are designed to achieve the guarantee of progress discussed in Appendix C. Figure 25 shows the region of attractions for the Stage 2 controller (red) and the Stage 4 controller (yellow), and the corresponding equilibrium points (depicted by the '+'). It is clear that an  $\varepsilon$ -ball of the equilibrium point for the Stage 2 controller is within the region of attraction for the Stage 4 controller.



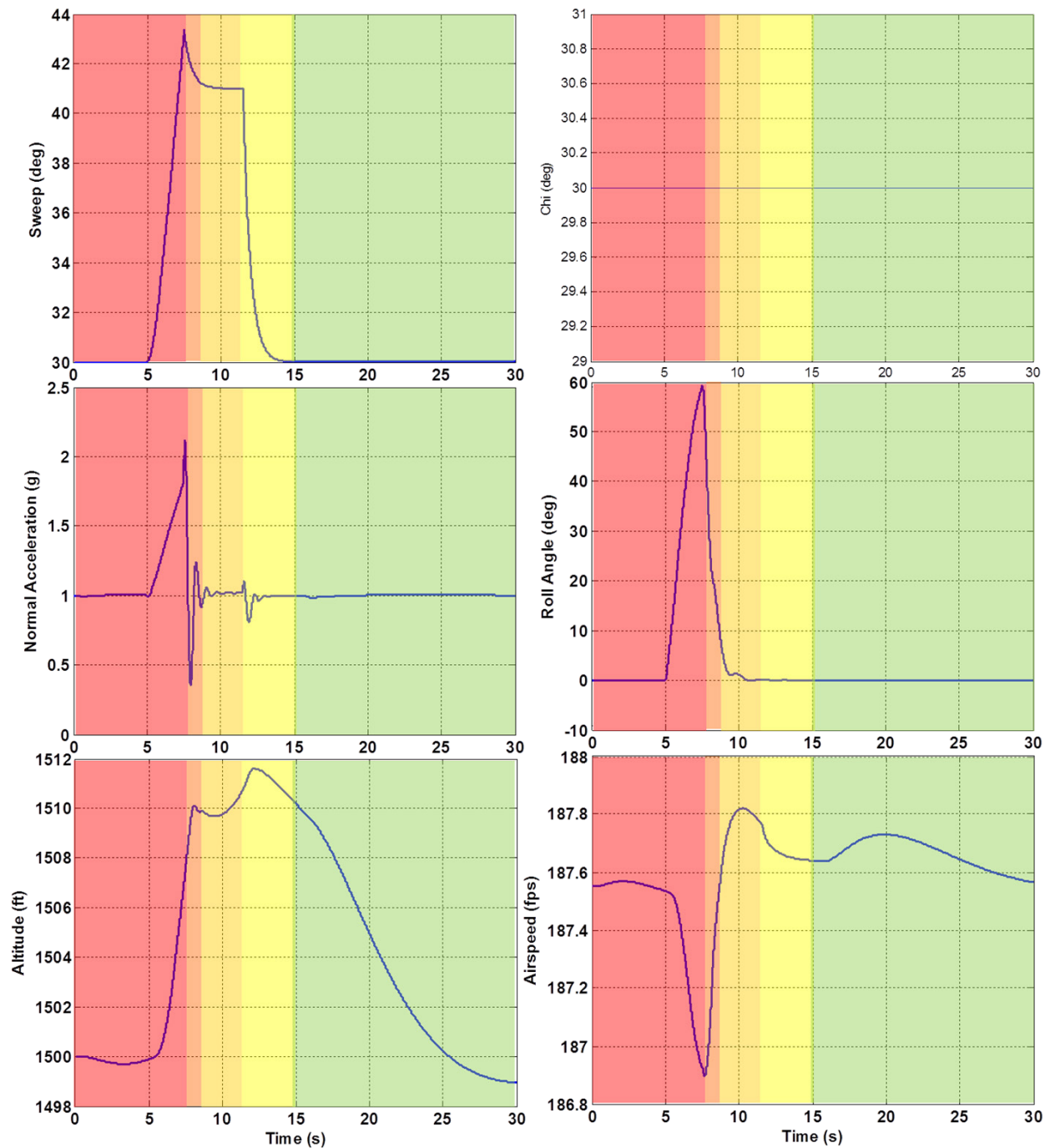
**Figure 25. Guarantee of Progress for RTA Demonstration: Mismanaged Morphing**

Figure 26 and Figure 27 show the simulation time histories for this experiment. In Figure 26, the advanced controller is active in the red shaded region. Stage 1 of the transition controller is active in the dark orange shaded region. Stage 2 of the transition controller is active in the light orange shaded region. Stage 3 of the transition controller is not required as the wing is already at the baseline value when the RTA switch is triggered. Stage 4 of the transition controller is active in the yellow shaded region. The baseline controller is active in the green shaded region. The switch from advanced to baseline takes about 8 seconds. As expected, airspeed does not change significantly during the transition.

We note here that the simulation model we used in these experiments was for the actual physical mock-up vehicle from the original morphing wing project. The physical mock up vehicle was a preliminary model that showcased the wing morphing mechanisms. As such, the time required to morph the wing from one planform to another took several seconds, depending on the required shape change, and this substantial amount of time is evident in these plots. It is presumed here that a production vehicle would have significantly faster morphing capabilities and the time from the RTA switch to the baseline configuration would be of the order of one second or less. A fast inner-loop reversion is desired as the inner-loop ignores commands from the guidance system while reverting (goes wings-level, goes to zero flight path, and then morphs to a baseline planform). The higher feedback levels should always take into account that the inner-loop RTA system could trigger a reversion at any time and these levels should never place the vehicle in a state that would not be able to allow for the time needed for inner-loop reversion (e.g., flying so close to an object, potentially requiring an immediate change in heading, which would have to wait until the inner-loop reversion completed).

Figure 27 shows time histories from a simulation run where the RTA switch is not triggered and the advanced controller continues to operate. At approximately ten seconds, a stall occurs and unstable flight ensues. This shows the benefit of RTA monitoring in which safe flight is maintained by properly switching to the reversionary controller, which mitigates the anomaly by

transitioning the state to a safe configuration. Without the RTA system, the incorrect morphing procedure leads to unstable flight and loss of the aircraft.



**Figure 26. Parameter Time Histories: Mismanaged Morphing with RTA Protection**

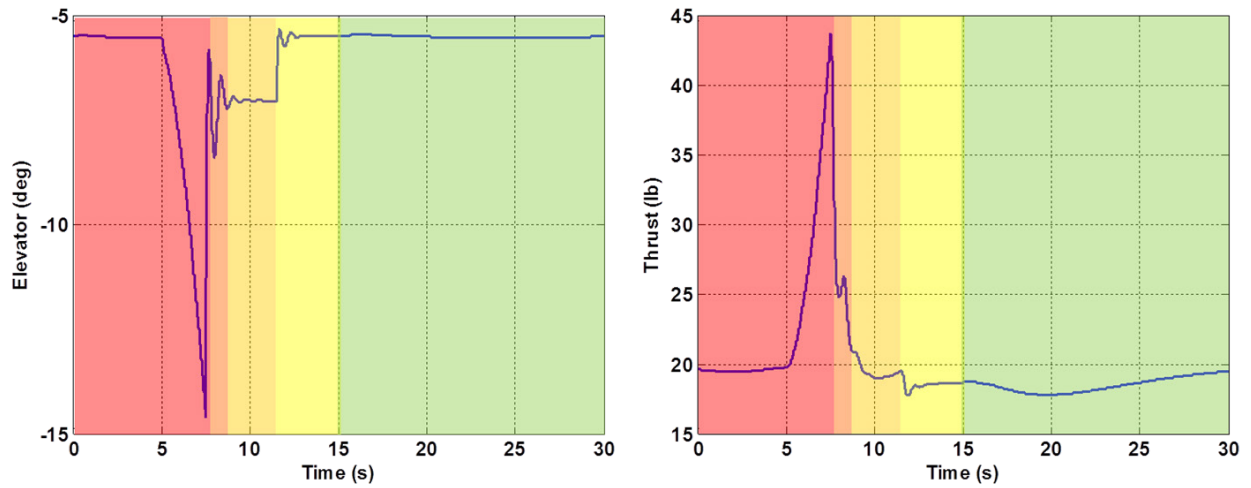


Figure (Continued)

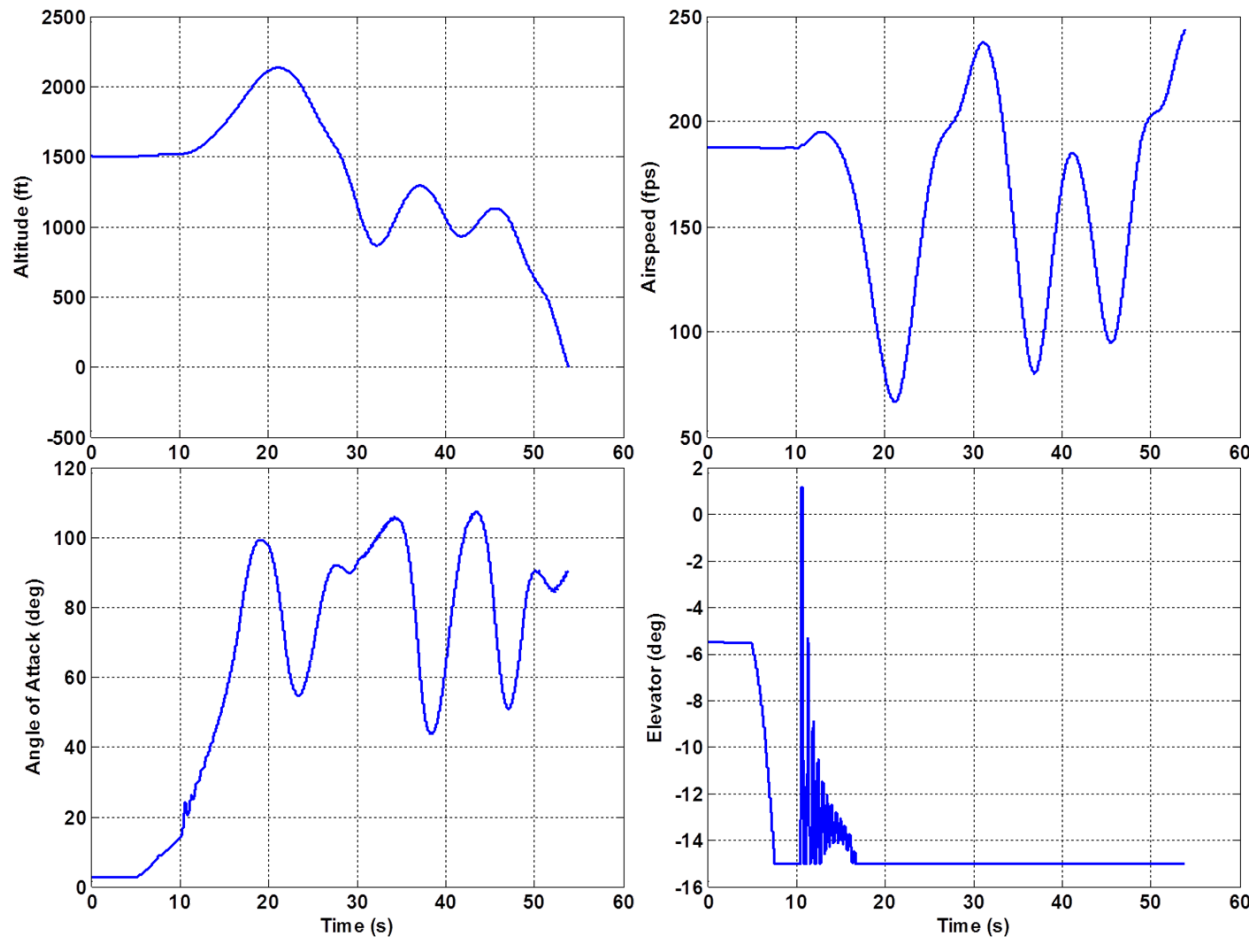


Figure 27. Parameter Time Histories: Mismanaged Morphing without RTA Protection

### 5.6.2 RTA Demonstration: Error in Advanced Controller

In this demonstration, an error in the advanced controller results in activation of the reversionary controller. The simulation is initialized in the Loiter configuration ( $S = 30, \chi = 30$ ). At 5 seconds, the advanced controller begins a transition to the High Speed Maneuver configuration ( $S = 50, \chi = 65$ ) along the black line in Figure 24. As the morph is initiated, a high bank turn is also commanded. At 10 seconds, an error in the parameter identification results in unexpected responses from the advanced controller. The parameter identification algorithm misidentifies the current effectiveness of the elevator; the advanced controller adapts to this change in elevator effectiveness by commanding a large increase in elevator. This results in an aggressive pitch down maneuver which puts the vehicle close to the negative AoA boundary. Proximity to this boundary triggers the RTA switch and the transition controller is enabled at approximately 11 seconds.

The transition controller follows the process outlined in Subsection 5.2.1:

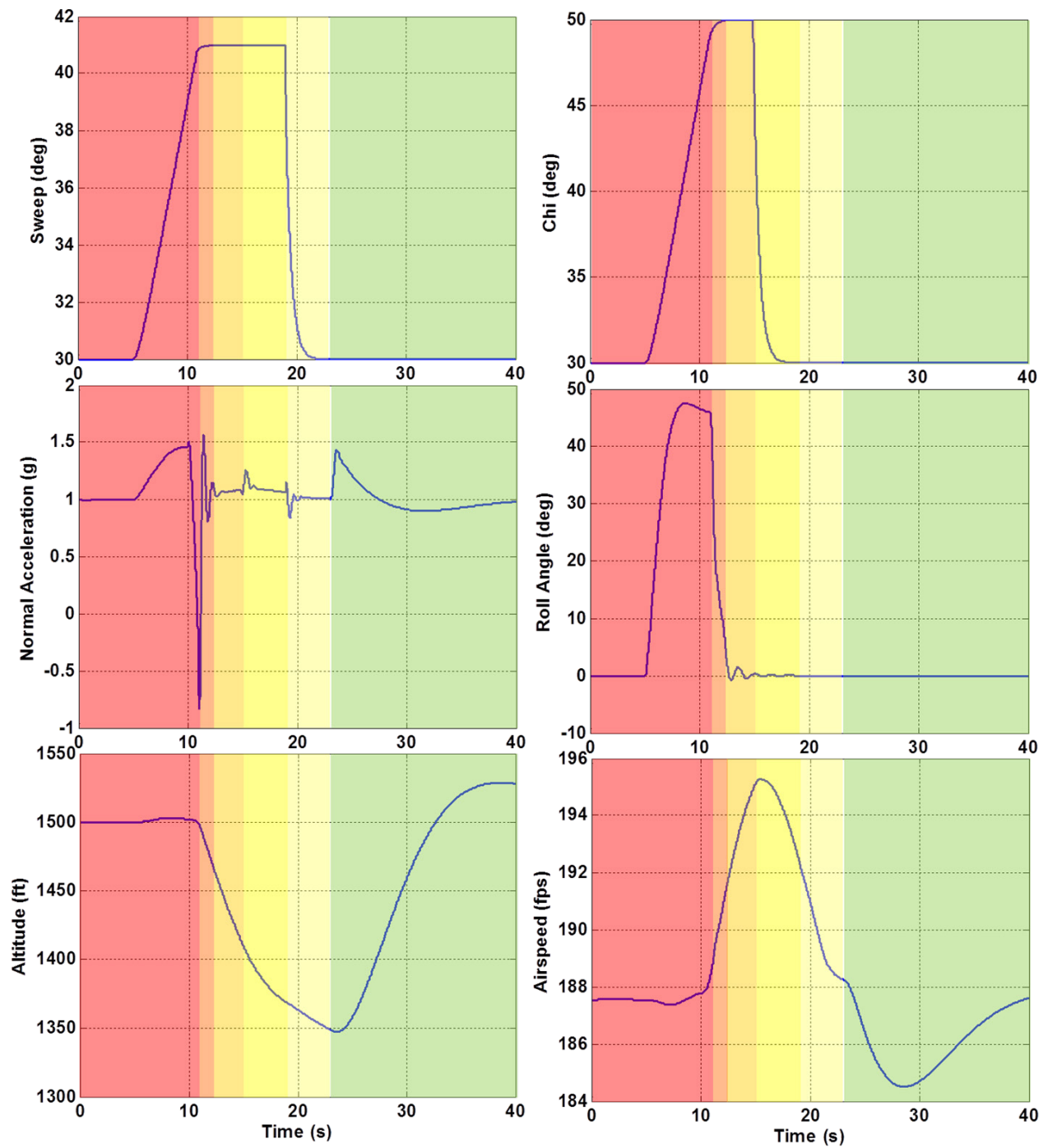
1. Stage 1: Go to wings level at the current wing configuration ( $S = 41, \chi = 50$ ).
2. Stage 2: Zero flight path at the current wing configuration ( $S = 41, \chi = 50$ ).
3. Stage 3: Transition to baseline chi while maintaining the current sweep ( $S = 41, \chi = 30$ ).
4. Stage 4: Transition to baseline sweep while maintaining the baseline chi ( $S = 30, \chi = 30$ ).

Control is passed to the baseline controller when the wing is back in the Loiter configuration. Again, the linear feedback laws that comprise the transition controller are designed to achieve the guarantee of progress discussed in Appendix C.

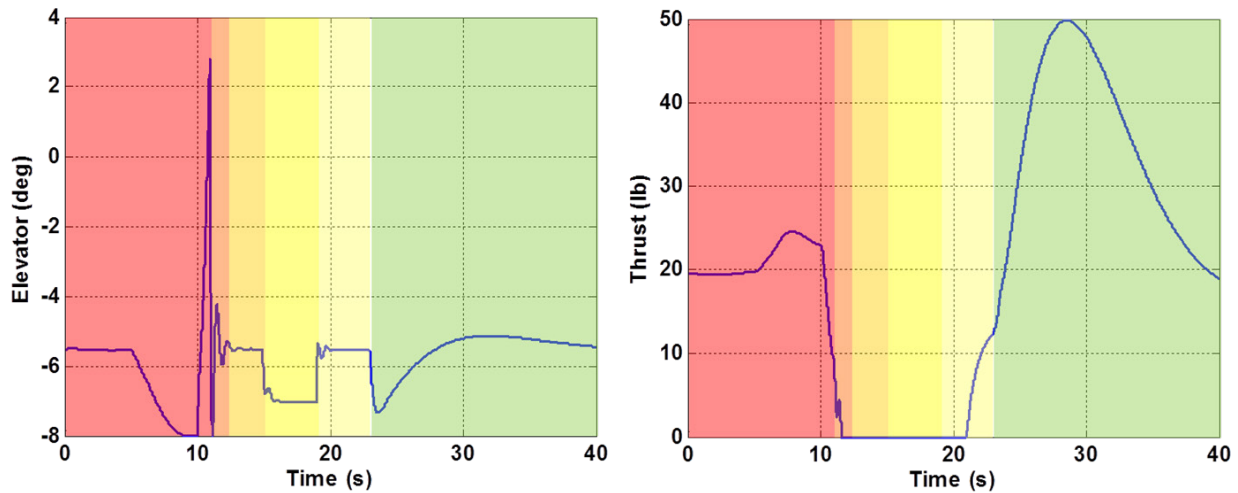
Figure 28 and Figure 29 show the simulation time histories for this case study. In Figure 28, the advanced controller is active in the red shaded region. Stage 1 of the transition controller is active in the dark orange shaded region. Stage 2 of the transition controller is active in the light orange shaded region. Stage 3 of the transition is active in the dark yellow shaded region. Stage 4 of the transition controller is active in the light yellow shaded region. The baseline controller is active in the green shaded region. The switch from advanced to baseline takes about 12 seconds. As expected, airspeed does not change significantly during the transition.

From these results, more specific A-G contracts can be defined for each transition controller in the multi-step transition process. Recall from Subsection 5.4.5, the assumptions for any inner-loop controller are the *rolled-up* assumptions of all the elements around the inner-loop feedback path. Let us define this set of assumptions as  $A_1$ .

First note that we notionally defined the desired set  $\mathcal{Q}_c$  to be when the vehicle achieve wings-level, zero flight path angle, or after the second transition controller completes its objective. From Figure 28, this is shown to be achieved within approximately 3 to 4 seconds from when the advanced controller is deactivated. Therefore, a typical value for  $T_c$  may be set to 5 or 6 seconds, for example.



**Figure 28. Parameter Time Histories: Error in Advanced Controller with RTA Protection**



**Figure (Continued)**

From Figure 28 and other simulation results, we can state the following A-G contracts:

For transition controller #1:

Assumptions:

- $A_1$ .

Guarantees:

- Roll angle commanded to zero in  $< 5$  seconds in smooth/stable manner (in this example, the guarantee is exceed, as the roll angle goes to zero in approximately 3 seconds),
- Stability maintained.

For transition controller #2:

Assumptions:

- $A_1$
- Bank angle  $< 3$  degrees

Guarantees:

- Stability maintained
- AoA brought to within 0.1 degree of equilibrium ( $= 3.5$  degrees) in less than 5 seconds.
- Pitch rate brought to within 1 degree/second of equilibrium ( $= 0$ ) in less than 5 seconds.

For transition controller #3:

Assumptions:

- $A_1$
- Bank angle  $< 3$  degrees
- Stable short period mode
- Within 0.1 deg of AoA  $= 3.5$  degrees
- Within 1 deg/sec of pitch rate  $= 0$  degrees/second.

Guarantees:

- Morphing angle Chi commanded to 30 degrees in less than 5 seconds.
- Stability maintained – move to new equilibrium point: AoA  $= 3.75$  degrees, pitch rate  $= 0$  degrees/second.

For transition controller #4:

Assumptions:

- A1
- Bank angle < 3 degrees
- Stable short period mode
- Within 0.1 degree of AoA = 3.75 degrees
- Within 1 degree/second of  $q = 0$  degrees/second
- Chi = 30 degrees.

Guarantees:

- Sweep angle commanded to 30 degrees in less than 5 seconds.
- Stability maintained – move to new equilibrium point: AoA = 2.6 degrees, pitch rate = 0 degrees/second.

For baseline controller:

Assumptions:

- A1
- Bank angle < 3 degrees
- Stable short period mode
- Within 0.1 degrees of AoA = 2.6 degrees.
- Within 1 degrees/second of pitch rate = 0 degrees/second
- Chi = 30 degrees
- Sweep = 30 degrees.

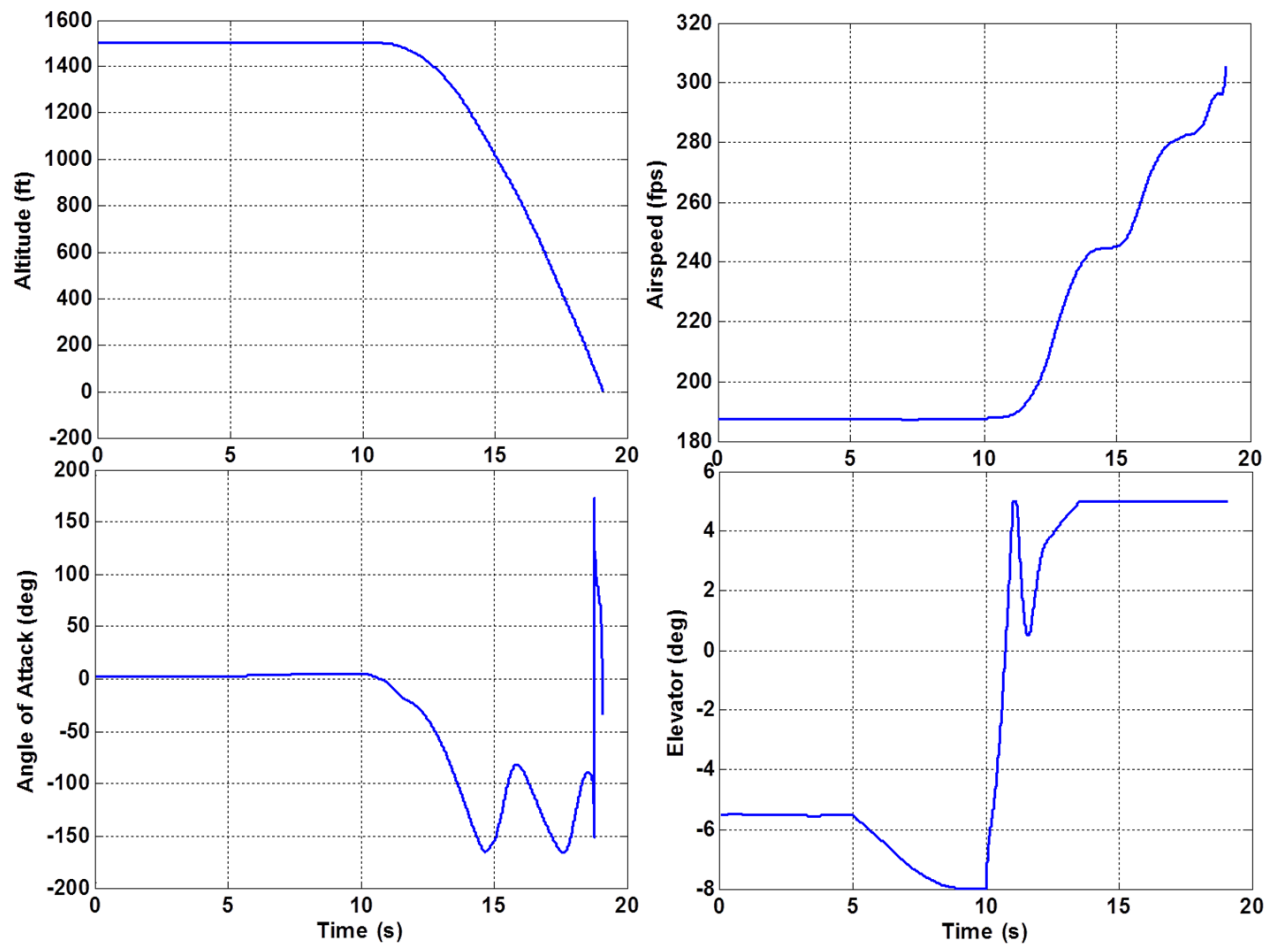
Guarantees:

- Stability maintained.
- Can control aircraft in loiter wing configuration to fly back to home airbase.
- Can switch to *landing* baseline controller coupled with autoland guidance on final approach.

Note how the assumptions for each successive transition controller build on the guarantees achieved by previous transition controller to the point that the state is in the operating region of the baseline controller, which safely recovers the aircraft.

Figure 29 shows time histories from a simulation run where the RTA switch is not triggered and the advanced controller continues to operate. The adaptation in the advanced controller results in holding an excessive negative AoA, resulting in eventual instability and loss of the aircraft.





**Figure 29. Parameter Time Histories: Error in Advanced Controller without RTA Protection**

## 6 RTA Protection Applied to Outer-Loop Guidance Systems

### 6.1 General Description of 3-DOF UAS Model used at the Guidance Level

The more detailed 6-DOF morphing wing model was the appropriate model to use at the inner-loop level because we investigate the aircraft's attitude states, which are a part of the 6-DOF model. However, for the guidance and higher level feedback loops, simulation experiments were performed using an analogous 3-DOF UAS model. Here, the aircraft's position states are the main focus, and its attitude dynamics are simplified to a point mass model. The 3-DOF UAS model and equations of motion are presented in Appendix D. Using this model, we constructed a multi-vehicle 3-DOF UAS simulation environment in Matlab/Simulink for numerical experiments and demonstrations.

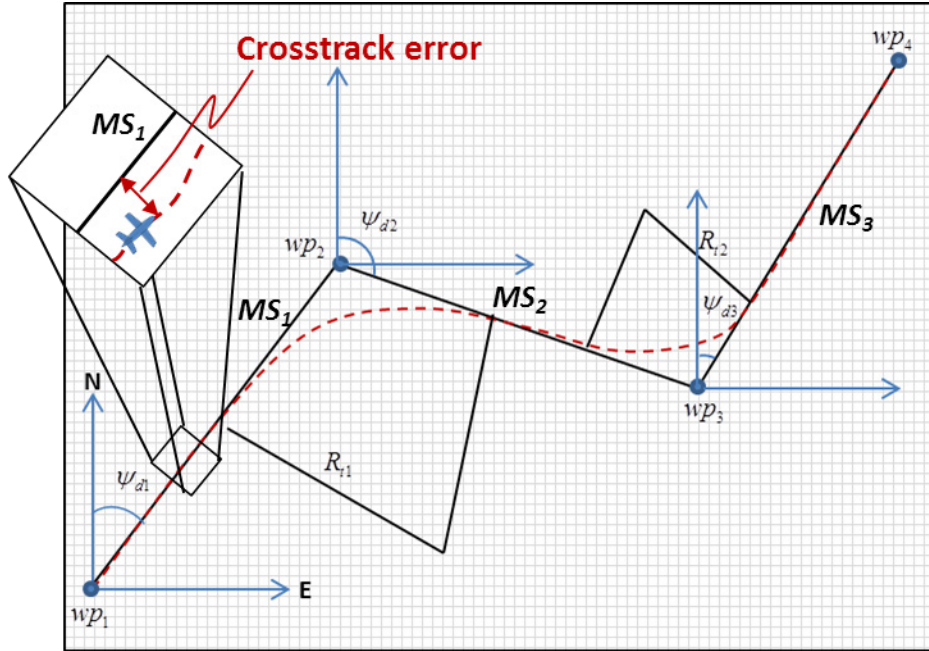
### 6.2 GLAW Functionality

The GLAW's main purpose is to follow a commanded path from the upstream FMS block. The two main functions of the guidance loop are:

1. Pass through altitude and airspeed: First, the FMS block delivers altitude and airspeed commands to the GLAW loop. In this project, we have defined the framework to be such that these commands are passed through the GLAW and delivered directly to the inner-loop CLAW. The inner-loop controller then follows the commanded altitude and airspeed through elevator and throttle feedback loops. Note that another common framework is to have the altitude be followed by a guidance law feedback loop and have the airspeed be followed by a separate throttle feedback loop (there is no aerospace industry standard). The input/output structure and feedback functionality defined in this project was simply chosen because the inner-loop controller from the past morphing wing project was designed to take in altitude and airspeed commands.
2. Lateral path following: The GLAW's main task is lateral path following. The guidance loop accepts {latitude, longitude} locations of waypoints generated by the FMS (or a {downrange, crossrange} location, defined in some local coordinate frame, {X,Y}, for example). The GLAW then solves for a bank angle command that should result in a heading that directs the vehicle to the waypoint location. This bank angle command is then delivered to the inner-loop, which generates the appropriate aileron and rudder commands to follow the bank angle command. Fundamentally the guidance system is generating steering commands and the inner-loop controller is performing the actual steering of the vehicle.

Details of standard waypoint following guidance are presented in Appendix D. This approach uses the past waypoint (or current location of the vehicle), the current waypoint that is being targeted, and the next waypoint beyond the current waypoint to form the proper geometry needed to follow the path defined by straight line segments between the waypoints. We define these straight line segments here as *mission segments* (MS). This geometry is illustrated in Figure 30. If the vehicle is flying on mission segment  $MS_1$ , then the waypoint 3-tuple {last, current, next} =  $\{wp_1, wp_2, wp_3\}$ . Once the vehicle has flown past  $wp_2$  and is flying on mission segment  $MS_2$ , then the waypoint 3-tuple is updated to be {last, current, next} =  $\{wp_2, wp_3, wp_4\}$ , and so on. Typically, waypoints are defined where there is a heading change in the commanded path. The amount of heading change defines the turn radius, which defines when the turn onto the next

mission segment should begin and is also used to calculate the commanded bank angle as a function of airspeed. More complex guidance schemes may also take into account the vehicle's turning performance capabilities so that it turns to the next commanded heading in a smooth, coordinated manner.



**Figure 30. Geometry for Standard Waypoint Following Guidance**

Figure 30 also shows how crosstrack error is defined, which is the lateral distance from where the vehicle should be located on the mission segment and where it actually is geometrically located. The other main objective of lateral guidance is to minimize crosstrack error. This is important for *spatial safety* of the vehicle. If the vehicle is not accurately tracking its commanded mission segments, then it runs the risk of collision with a neighboring fleetmate or other objects. The topic of spatial safety will be revisited later in this chapter.

### 6.2.1 Difference between Advanced and Reversionary GLAWs

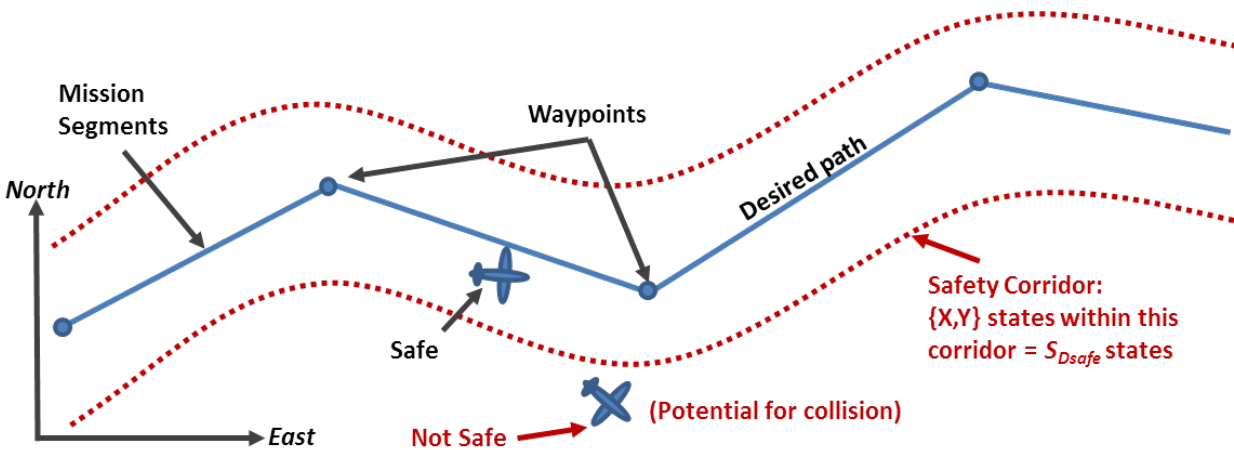
Since we have defined a modular architecture, the guidance system can be one of three options: 1) an industry standard, design time assured GLAW, (DTA GLAW); 2) an RTA protected GLAW running in advanced mode, (AGLAW); or 3) an RTA protected GLAW running in reversionary mode, (RGLAW). Each of these three options must therefore have the same input/output mapping. That is, they each a) pass through altitude and airspeed commands from the FMS to the CLAW, and b) take in  $\{X,Y\}$  waypoint locations as they become available from the FMS and calculate a bank angle command that is then delivered to the CLAW.

We assume here that the RGLAW is an industry standard waypoint follower that uses the waypoint 3-tuple  $\{\text{last, current, next}\}$  to generate the current heading and solve for a bank angle command to make a coordinated turn when appropriate. The RGLAW also uses a classical control design approach to generate incremental bank angle commands as a function of the current crosstrack error.

The AGLAW can take on many forms and several options can be found in the literature, e.g., [Cooper 2014]. We assume here that the AGLAW uses (at least) the waypoint 3-tuple as in the RGLAW. However, if additional future waypoint locations are available from the FMS, it may work with more than these three. We assume the AGLAW converts the waypoint locations to a continuous path and may use some advanced optimization or adaptation scheme to improve on the path following performance. In [Cooper 2014], an advanced guidance scheme is presented that uses L1 adaptation to provide better tracking of a continuous path under unforeseen disturbances. Since the feedback law in this approach depends on current environmental conditions, it would most likely be difficult to certify with current V&V methods. Nevertheless, the AGLAW delivers a bank angle command that is the solution of this advanced algorithm. It may therefore not be the same value as what would be calculated by the RGLAW.

### 6.3 Type Safety at Outer-Loop GLAW Level

Again we define the set  $S_{Dsafe}$  as the set of states in which it has been determined (through analysis, simulation studies, etc.) that the plant or system can operate safely and correctly. Recall that at the inner-loop level, this safety involves the structural, aerodynamic and attitude dynamic properties of the vehicle. However, at the outer-loop GLAW level, safety only involves spatial safety. We define  $S_{Dsafe}$  as those position states that lie within a pre-defined safety corridor, through which the aircraft is commanded to fly. This is depicted in Figure 31. Safety corridors are typically defined in the terminal areas for commercial aircraft to manage the air traffic flow into and out of an airport. We adopt that concept here for Air Force UAS mission applications.



**Figure 31. The Concept of a Safety Corridor**

As long as the vehicle remains within the safety corridor, then it is guaranteed to be safe from collision with ground objects (there may be margin built into the definition of the safety corridor, but the mission planners define this as the *never-go-beyond line*). The width of the corridor could be defined as a function of the particular mission scenario or the airspace geometry for that mission. For example, flight at higher altitudes or over open terrain can have large safety corridors. However, flight through forests, mountainous regions, urban terrains, etc., with many hazardous objects may require very *tight* corridors with little margin for error. We assume the

corridor width may be set to be a constant during pre-mission planning, or it may be dynamically reset during the mission due to changes in the airspace characteristics or other mission considerations. We define here that the safety corridor path and width (potentially as a function of time and/or location) are planned out by the higher level FMS loop. The geometric mapping of the safety corridor boundary is key to defining the RTA safety check at the GLAW level, so we define here that this information must be passed to the GLAW RTA monitor during flight.

With this, we can now formally define safety levels that are used to determine the switching condition protocol for an RTA protected outer-loop guidance system. We will use the capital letter ‘G’ to denote that these safety definitions are for the guidance loop level.

**Definition 6.** Outer-Loop Guidance Type Safety - A point  $x_0$  in the state space  $S$  is:

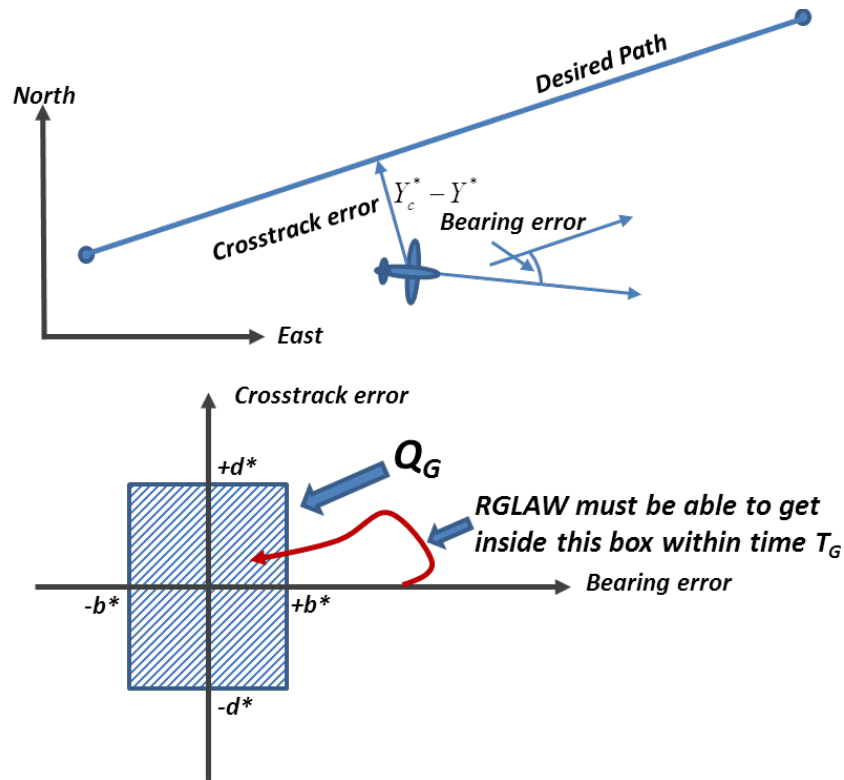
- **GType I Safe** if that point lies inside  $S_{Dsafe}$  (= the safety corridor)
- **GType II with set  $Q_G$  and time  $T_G$  Safe** if all of the following hold:
  - a) Point  $x_0$  is **GType I safe**,
  - b) Upon switching to the reversionary **GLAW**, the state trajectory can converge to at least one point in a *desired* set  $Q_G$  within a given *desired* time  $T_G > 0$ ,
  - c) The state trajectory from the point of switching to the reversionary system to the point of reaching the set  $Q_G$  is entirely contained within the **GType I safe** region.
- **GType III with Period  $\tau_G$  Safe** if all of the following hold:
  - a) Point  $x_0$  is **GType II safe**,
  - b) Every possible output of the advanced system for a time period  $\tau_G$  results in a state trajectory entirely contained within the **GType II safe** region.

Since the outer-loop dynamics operate at a very high bandwidth, the period  $\tau_G$  will be relatively small in value (of the order of 0.1 seconds, for example). Therefore, the RTA monitor should be updated at least as fast as the outer-loop guidance system update.

To define the desired region  $Q_G$ , consider that the two main parameters that define good path tracking are crosstrack error and bearing error (the error between the desired heading and the actual heading). Therefore, we define  $Q_G$  to be the region in state space in which:

$$|\text{Crosstrack error}| < d^* \cap |\text{Bearing error}| < b^* \quad (9)$$

Where  $d^*$  and  $b^*$  define desired values for crosstrack and bearing errors, respectively. This is depicted in Figure 32. The time  $T_G$  is defined as a time in which the RGLAW can be reasonably expected to achieve  $Q_G$ . The desired region can be defined in other ways, but this definition is a straightforward option based on fundamental characteristics of path following geometry.



**Figure 32. Definition of Desired Region  $Q_G$**

The GType I – III boundaries are geometrically depicted in Figure 33. As defined, the GType I boundary is the boundary of the safety corridor. It can be seen that the vehicle has both a crosstrack error and a bearing error. If the heading is turned away from the desired path at the time the RTA switches to the RGLAW, then a path transient (or overshoot) will occur that first leads the vehicle away from the desired path before turning it toward the desired region  $Q_G$ . Therefore, the GType III switching condition must be defined such that this path transient never goes beyond the GType I boundary (the safety corridor).

Also, note that Figure 33 shows that the GType II and III boundaries are not constant distances away from the safety corridor, nor are they straight lines. Rather, these will be complex functions of many states that can influence the size of the path transient, such as crosstrack error, bearing error, airspeed, bank angle, dynamic pressure, etc. For example, if the bearing error is smaller or such that the heading is turned toward the desired path, then the allowable crosstrack error can be larger. Vice versa, if the crosstrack error is small, then the allowable bearing error can be larger. Higher airspeeds will reduce the allowable crosstrack error, whereas lower airspeeds will increase the allowable crosstrack error because the vehicle will have more turning capability at lower velocities. A numerical example that shows how these boundaries can be constructed is given at the end of this chapter.

The concept of a safety corridor is ideal for a single UAS. However, for a fleet of UASs, each will have to have a safety corridor that does not conflict with its neighboring fleetmates. For missions through urban terrain, there may be *bottlenecks* which do not allow for all corridors to fit through without encroaching on each other spatially. In that case, the vehicles will need to be

deconflicted temporally, as for example, reconfiguring into a line with one vehicle following another. In this case, the concept of merged corridors can be obscure. Instead, for multi-vehicle missions, let us borrow an idea that is common in collision avoidance approaches, [Cooper 2014]. Instead of managing safety corridors, let us define *separation volumes* around each vehicle in the fleet. Such volumes are shown in Figure 34. In air traffic collision avoidance, the FAA has decreed that each vehicle should maintain a well clear distance from other aircraft. This defines what is termed the self-separation volume. Under emergency conditions, each vehicle must maintain a minimum separation distance, which defines what is termed the collision volume. If this volume is breached, there is a high probability of collision.

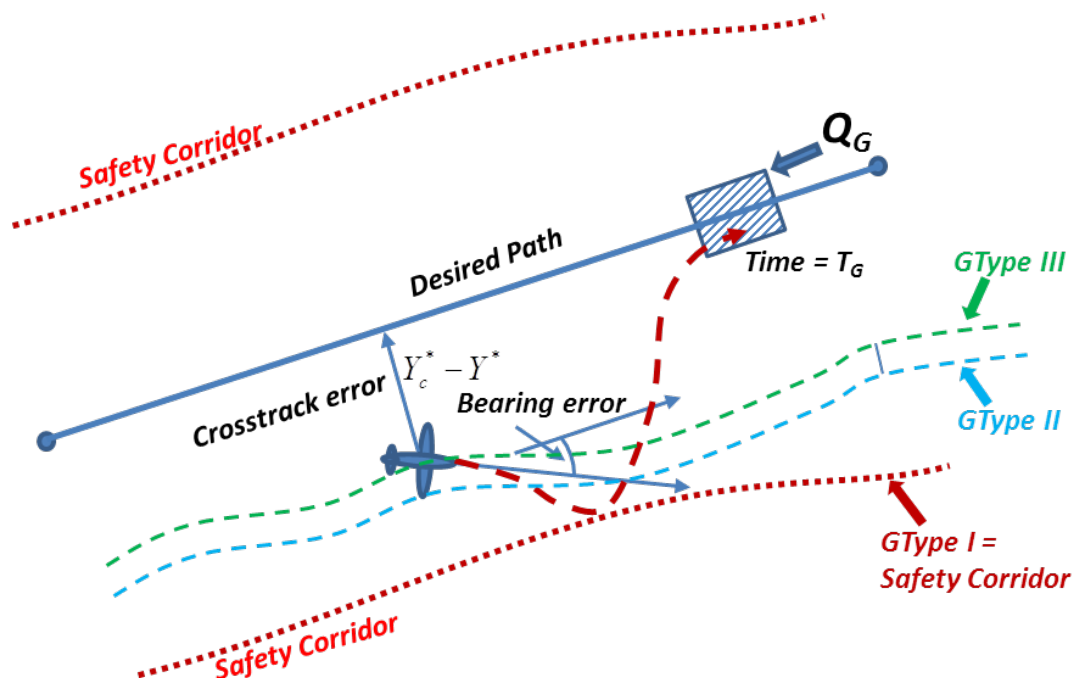


Figure 33. Geometric Interpretation of GType I – III Boundaries

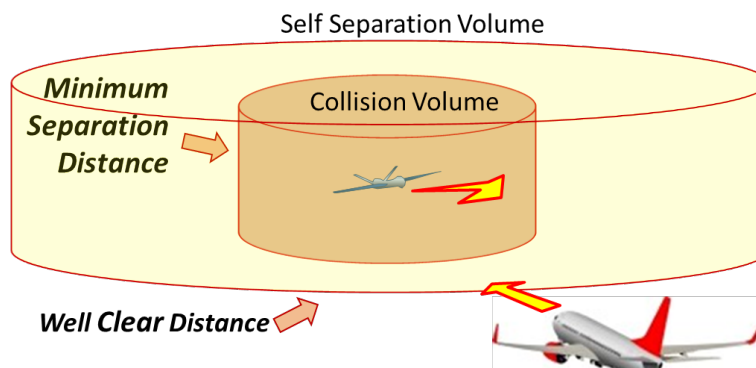


Figure 34. Separation Volumes used in Collision Avoidance Schemes

For our challenge problem, we define the general concept of a *required separation volume* (RSV) around each vehicle in the fleet. The radius and height of this cylindrical volume can

dynamically change throughout the mission depending on current airspace conditions (e.g., open terrain, urban terrain, etc.); the type of airborne threat (e.g., cooperating fleetmate, non-cooperating intruder, unmanned vehicle, high value, piloted vehicle, fast or slow closing speed, etc.); or the type of required ground avoidance (e.g., no-fly zone boundary, physical object, such as a building or tower, enemy anti-aircraft battery, etc.). The more risk perceived by the threat will require a larger RSV. We define here that the FMS continually manages/adapts the size of the RSV for each vehicle depending on the current mission conditions. However, it is the task of the GLAW to maintain the vehicle safely within the RSV. Figure 35 depicts the equivalence of the GType I, II and III boundaries for the safety corridor geometry and for the RSV geometry.

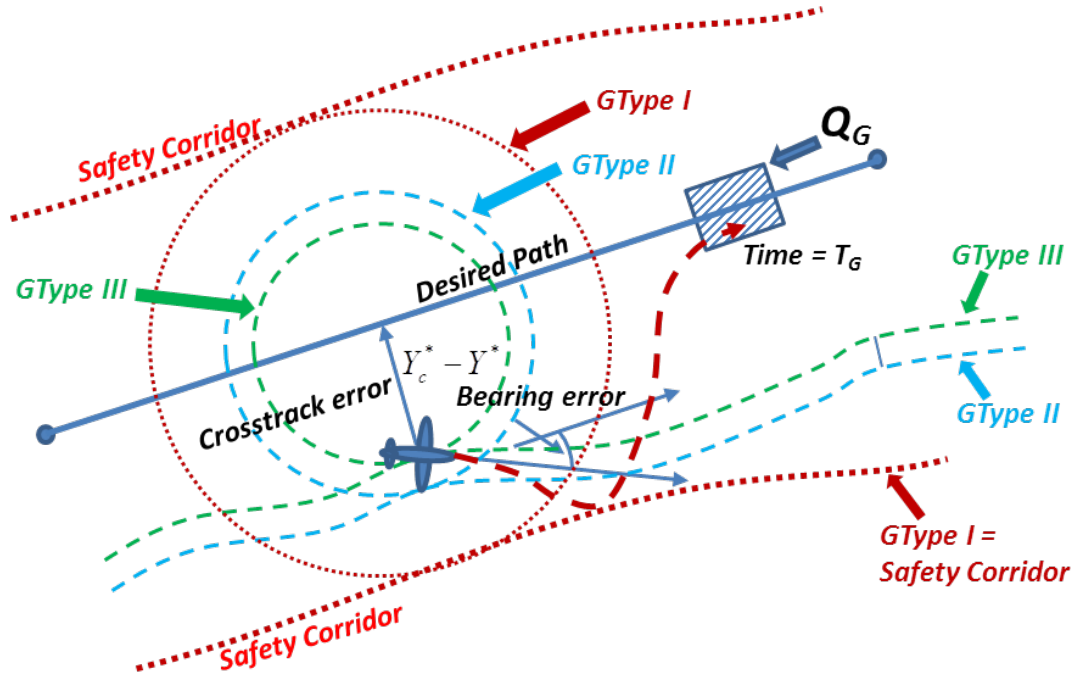


Figure 35. Equivalence of GType I – III Boundaries between Safety Corridor and RSVs

#### 6.4 A-G Contracts at Outer-Loop GLAW Level

We now revisit the compositional reasoning approach. To illustrate this reasoning, consider that an aircraft is experiencing large crosstrack errors. These path tracking errors could be a result of an error at the inner loop level. The aileron control law will be a function of the bank angle error, as:

$$\bar{e} = \begin{bmatrix} H_c - H \\ \phi_c - \phi \\ V_c - V \end{bmatrix} \Rightarrow e_\phi = \phi_c - \phi, \quad \delta_a = K(X)(\phi_c - \phi) \quad (10)$$

whether the inner-loop controller is advanced, reversionary or design-time assured. If this control law is incorrectly designed or incorrectly coded, this can lead to growing crosstrack errors. However, by compositional reasoning, when building A-G contracts at GLAW level, we postulate that the A-G contracts at CLAW level hold. This is the fundamental basis for this



reasoning. Therefore, we have postulated that there are no errors present in the inner-loop controller (because it is either design-time assured or it is runtime protected). Therefore, as we construct A-G contracts at GLAW level, any path tracking errors will only be due errors at the GLAW level, and these will be addressed through reasoning about how the RTA protection will mitigate such conditions.

Figure 36 illustrates how the inner-loop level (see Figure 23) is collapsed into the composite *black box* that is certified to be safe and correctly operating at all times (by the compositional reasoning logic).

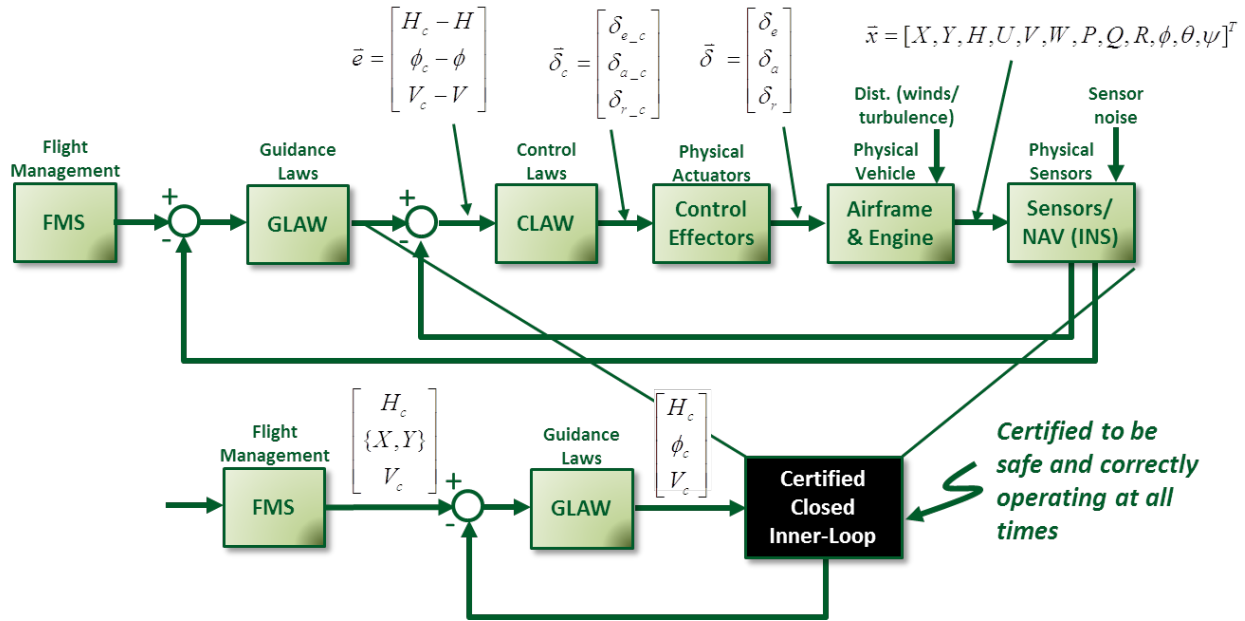


Figure 36. Development of A-G Contracts at the Outer-Loop GLAW Level

#### 6.4.1 Assumptions on Inputs from the Upstream FMS

Inputs to the GLAW feedback loop are commanded altitude, waypoint  $\{X, Y\}$  locations, and commanded airspeed. Since the GLAW simply passes the commanded altitude and airspeed through to the inner-loop, the assumptions on these commands remain the same as discussed in Subsection 5.4.1 (see Eq. (1)). For the waypoint locations, we assume these adhere to standardized guidelines, such as:

- Positioning of waypoints shall not require climb, descent or turning performance beyond vehicle's capabilities
- Waypoints shall not be spaced so close together that vehicle cannot properly follow them
- Waypoints shall not be placed in hazardous locations (near objects, terrain that can lead to collisions, etc.)
- Waypoints shall not be placed in restricted airspace or no fly zones

Similar industry standard guidelines exist, such as from Terminal Instrument Procedures (known as TERPS).

#### 6.4.2 A-G Contracts for Certified Closed Inner-Loop System

The assumptions on the inputs to the closed inner-loop system have been covered in Subsection 5.4.1, and again involve allowable values on the commands, their rates and frequency content. As long as these assumptions hold, then the guarantees are the rolled-up guarantees of the inner-loop system, namely:

- i. Vehicle is stable in attitude
- ii. Vehicle is structurally intact
- iii. Airflow attached, etc.
- iv. Vehicle closely follows altitude, airspeed and bank angle commands by some predefined metrics.

#### 6.4.3 A-G Contracts for the Outer-Loop Guidance System

We will summarize the A-G contracts that must hold for the outer-loop GLAW, be it an advanced, reversionary, or DTA GLAW (that is, in any of these cases, the A-G contracts must hold, although specific values or measures for, for example, performance requirements may be different for different GLAWs). Since the GLAW is designed with the fundamental assumptions made on the rest of the elements shown in Figure 36, the assumptions here are that all A-G contracts for all the other elements around the feedback loop hold. That is,

- i. A-G contracts hold for the certified closed inner-loop system
- ii. Assumptions on FMS commands hold

As long as these assumptions hold, then the guarantees are that the GLAW:

- i. Maintains translational stability of the airframe for all time,
- ii. Achieves the required path tracking performance for all time,
- iii. Stability and performance robustness is maintained for all time, and
- iv. Stated assumptions on command inputs to the closed inner-loop system hold for all time.

For path tracking performance, this may be ascertained by certain defined characteristics of the crosstrack error, such as that a norm bound (e.g., peak value) holds, or a settling time is achieved, etc. (again, the measures will depend on the specific GLAW design approach).

Stability and performance robustness guarantees will be with respect to specified measures of plant modeling uncertainties, plant disturbances (e.g., turbulence magnitude) and sensor noise characteristics. Last, stability, performance and robustness are all guaranteed while delivering bank angle commands to the control effectors that do not exceed rate or deflection limits and have valid frequency content.

#### 6.4.4 A-G Contracts for the GLAW RTA System

For the reversionary GLAW, the A-G contracts will be equivalent in structure to the GLAW in general. However, specifically, the reversionary GLAW also guarantees that at least GType I Safety holds for all time (the vehicle will always remain within its RSV or safety corridor), and that the desired region  $\mathcal{Q}_G$  is achievable within time  $T_G$  when it is first activated by the RTA monitor and switch mechanism.

As with the inner-loop, the reversionary GLAW's guaranteed path following performance characteristics may be less, by some measure, than the advanced GLAW. If this is the case, then if the RTA monitor commands a switch to the reversionary GLAW, this will change the GLAW's A-G contract at the outer-loop level. This change or reduction in the outer-loop GLAW's performance capabilities must be communicated to all feedback levels, as this can affect how well the aircraft can stay within its RSV or safety corridor.

For the RTA monitor and switch mechanism block, the main assumptions are that the information input to this block is valid (correct to within some accuracy tolerance) and that it at least starts its operation while the system is within the GType III safety envelope. That is, the RTA system cannot make any guarantees of safety if it does not start monitoring the system state until after it has crossed the GType III safety boundary. This is not a limiting assumption since nominally the RTA system will be activated at the time the UAS vehicle is launched at mission start and presumably it would not be launched in unsafe conditions.

As long as the above assumptions hold, the guarantees for the RTA monitor and switch mechanism are that:

- i. Loss of GType III safety is always detected at first occurrence, while the system is in the GType II safe region, and
- ii. The reversionary GLAW is activated when loss of GType III safety is detected.

## **6.5 Outer-Loop GLAW RTA Checks and Switching Conditions**

Recall, the general checks the RTA monitor must perform were listed in Subsection 3.4.1. We can now specify these checks for the outer-loop feedback level.

1. Safety check: At each update to the outer-loop RTA monitor, it checks the system state to determine if it has left the GType III safe region. If so, the advanced GLAW is shut down and guidance is switched to the reversionary GLAW.
2. Output/environment check: The outer-loop RTA monitor checks that the bank angle commands generated by the advanced GLAW do not violate any constraints imposed on the closed inner-loop system. Again, this will involve rate and magnitude limits on bank angle command, as well as its frequency content.
3. Performance check: The RTA monitor checks that the advanced GLAW is achieving its minimum required path tracking performance, which will involve crosstrack and bearing error, and possibly other measures as well. Again, these checks may include transient response specifications, such as rise time for commanded step inputs, or settling times for disturbance rejection measures. Steady state norm bounds on tracking errors are another example of a common performance measure.
4. Input/environment check: The RTA monitor should also check that inputs to the outer-loop feedback level from the FMS block do not violate any constraints imposed on the inputs. Such constraints will include magnitude and rate limits, frequency content, and possibly other statistical measures on the altitude and airspeed commands. Also, standard checks on the relative waypoint locations should also be checked to ensure the vehicle can physically follow the waypoint selections.

5. System hardware health check: As with the inner-loop, assumptions on hardware and information integrity should be checked and mitigation strategies coordinated with an IVHM or RM system.

## 6.6 Experimental Results

We performed preliminary studies in developing models for the GType I – III boundaries using our 3-DOF UAS simulation model. For simplicity, we focused on the safety corridor concept, as this is easier to visualize for a single UAS vehicle. Figure 37 presents the distances that need to be calculated for determining the GType III switching condition. It is assumed the coordinates through space are known for the desired path and safety corridor boundaries. At the current time, the crosstrack error ( $xte$ ) is:

$$xte = Y_c^* - Y^* \quad (11)$$

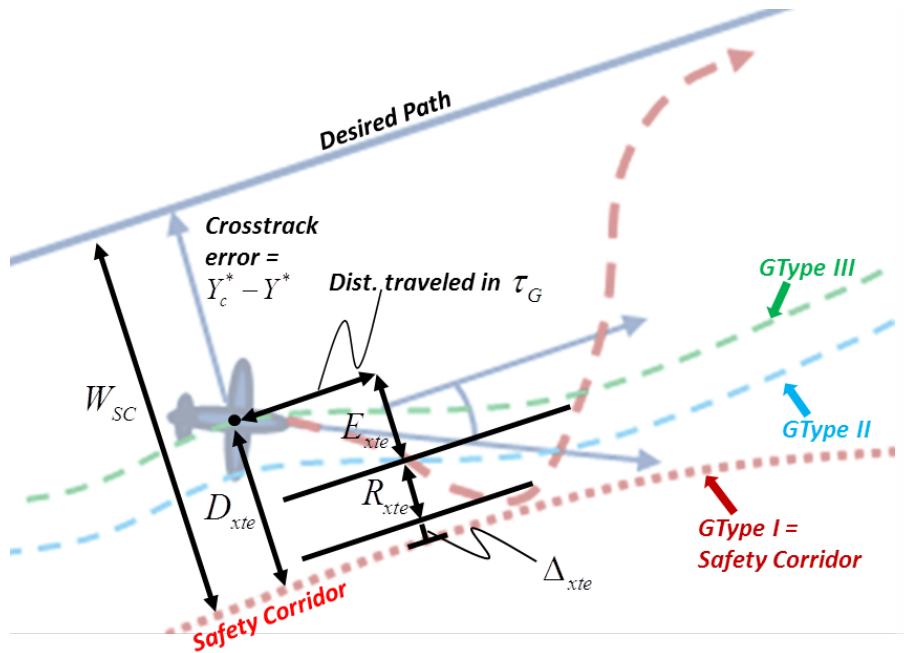
Using this, the distance the vehicle's location is from the safety corridor boundary is defined as  $D_{xte}$  and can be determined from the current width of the safety corridor,  $W_{SC}$ ,

$$D_{xte} = W_{SC} - (Y_c^* - Y^*) \quad (12)$$

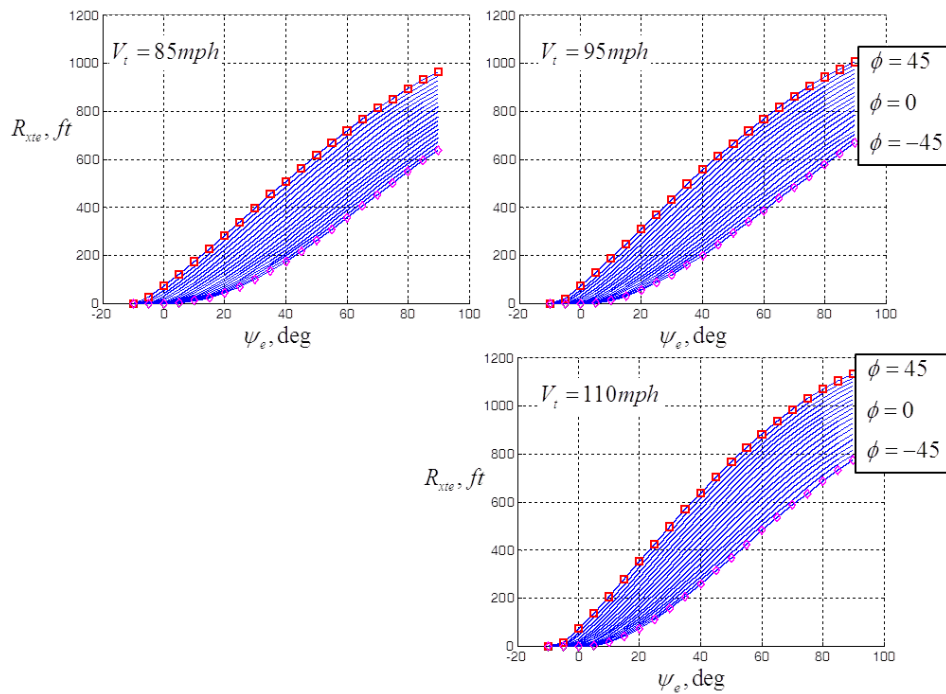
As shown in the figure, we assume a bearing error that drives the vehicle further from the desired path. The distance the vehicle will travel in time  $\tau_G$  (the RTA monitor update rate) extends the crosstrack error by an amount  $E_{xte}$ . At the time  $\tau_G$  in the future then, the vehicle will be a distance  $R_{xte} + \Delta_{xte}$  from the safety corridor boundary (GType I), where  $\Delta_{xte}$  is an added distance margin to account for winds/turbulence and other modeling uncertainties. The quantity  $R_{xte}$  is defined as the crosstrack distance traveled by the vehicle before the heading is turned back toward the desired path. This quantity is a function of the reversionary GLAW and the physical maneuvering capabilities of the closed inner-loop system as well as the current state. That is, it is the crosstrack overshoot experienced by the vehicle once the RGLAW has taken over the guidance function. The three states that will influence  $R_{xte}$  the greatest are current bearing error,  $\psi_e$ , current bank angle,  $\phi$ , and current airspeed,  $V_t$ . We focused only on these states and defined:

$$R_{xte} = f_R(\psi_e, \phi, V_t) \quad (13)$$

From the UAS simulation, we generated empirical results for this function, shown in Figure 38 for three different airspeeds, bank angle ranging from -45 to +45 degrees, and bearing error ranging from -10 to 90 degrees. As expected, Figure 38 shows that as the crosstrack overshoot increases in magnitude as the airspeed increases and increases in magnitude as the bearing error increases. A positive bank angle indicates that the aircraft is initially banked in the direction that drives the vehicle away from the desired path, whereas a negative bank angle means the vehicle is banked in the direction of the desired path. Hence, the more positive bank angle, the greater the crosstrack overshoot and vice versa, the more negative the bank angle the less the crosstrack overshoot.



**Figure 37. Defined Distances for Calculating GType III Switching Condition**



**Figure 38. Crosstrack Overshoot due to RGLAW Reversion**

The distance between the GType III and GType II boundaries is defined by the quantity  $E_{xte}$ . This quantity is determined empirically as the worst case additional crosstrack error that vehicle can achieve in  $\tau_G$  seconds. That is,

$$E_{xte} = f_E(\psi_e + \Delta\psi_{e\_worst}(\tau), \phi + \Delta\phi_{worst}(\tau), V_t + \Delta V_{t\_worst}(\tau)) \quad (14)$$

where the worst case additional amount of bearing error, bank angle (in the wrong direction) and airspeed can get in  $\tau_G$  seconds is added to their current respective quantities. We can equivalently express this as

$$E_{xte}(\psi_{e\_worst}, \phi_{worst}, V_{t\_worst}) = R_{xte}(\psi_{e\_worst}, \phi_{worst}, V_{t\_worst}) + D_\tau(\tau) \quad (15)$$

where we use the same simulation function as was used to calculate  $R_{xte}$ , but now with the worst case quantities and adding in the distance the vehicle can travel in  $\tau_G$  seconds, which can be calculated using Euler integration and simple geometric relations,

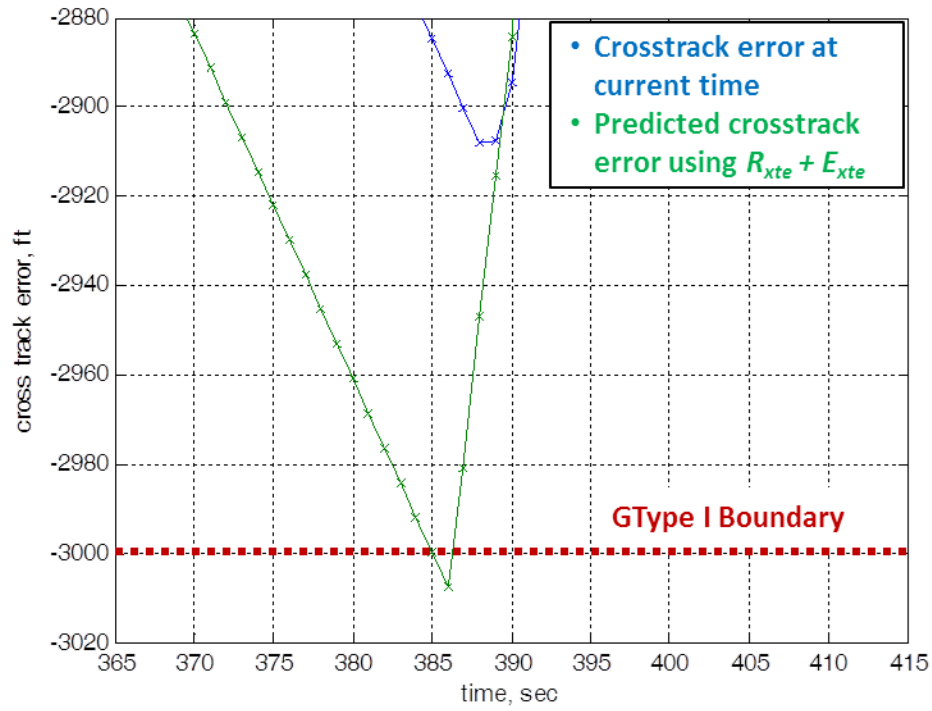
$$D_\tau(\tau) = \left[ \frac{V_t + (V_t + \Delta V_{t\_worst}(\tau))}{2} \tau \right] \sin(\psi_e + \Delta\psi_{e\_worst}(\tau)) \quad (16)$$

From Figure 37, the GType III switching condition check is then:

$$\text{Is: } D_{xte} - (R_{xte} + \Delta_{xte} + E_{xte}) \leq 0? \quad (17)$$

Simulation runs were performed introducing a seeded fault into the guidance law that caused the vehicle to drift from its desired path toward a defined safety corridor boundary. Figure 39 shows an example run which indicates the benefit of switching at the GType III boundary. In the figure, the blue line represents the crosstrack error at the current time for the vehicle. The green line plots the predicted future crosstrack error if the RGLAW is activated at the current time using the switching condition given in Eq. (17). It can be seen that between 385 and 386 seconds, the predicted crosstrack error violates the GType I boundary. This triggers the switch to the RGLAW, which begins to correct the crosstrack error. Between 388 and 389 seconds, the actual or current crosstrack error (blue line) begins to decrease (the vehicle begins to arrest the crosstrack error). The maximum crosstrack magnitude at this time is approximately 2910 ft, which gives a crosstrack margin of approximately 90 ft. The RTA switching condition can be made more or less aggressive (more or less margin at the switching condition) by adjusting the margin quantity,  $\Delta_{xte}$ .

These initial studies were a first look at the how one could construct the GType III switching condition using empirical results from simulation studies. Clearly, a full-up, detailed design of a real-world application of RTA monitoring at the GLAW level would involve more complex, higher fidelity simulation studies, investigating many additional states that can influence the crosstrack overshoot experienced by the vehicle. As with the inner-loop, here too the GType III switching condition boundary will be a complex hypersurface in state space. If the construction of this boundary is highly trusted, this will allow reduced required margins, but this comes at a cost of extensive labor manhours, and simulation/analysis processes.



**Figure 39. Example Experimental Result for GLAW RTA Protection**

## 7 RTA Protection Applied to Flight Management Systems

### 7.1 General Description of the FMS Functionality

The FMS block will notionally be made up of a number of subsystems, each performing a certain management task. In commercial aircraft, one of the main functions of the FMS is to manage the navigation of the vehicle and the waypoints along its assigned route. For our UAS platforms, the FMS would be tasked with managing the vehicle's payload sensor suite, for example, ensuring sensor modes (e.g., surveillance mode, situational awareness mode, etc.) are properly assigned at the correct time during the mission. Another FMS task may be to stow or deploy the landing gear or release stores, for example. These and other vehicle management functions can certainly have safety implications if they are not correctly executed, requiring RTA monitoring if design-time certification cannot be achieved. To address all the complexity of the myriad functions within the FMS block was beyond the scope of this program. Instead, we focused only on the navigation function. For this function alone, the input/output structure of the FMS is:

#### 1. Inputs from the MPS:

- a. A series of fleet rendezvous point (RP) locations in some coordinate frame, such as {Lat., Long., Altitude} or some equivalent local frame as {X, Y, H},
- b. A timing plan that specifies required arrival times to the set of RPs,
- c. Terrain, airspace and/or theater information with locations of hazards, no-fly zones, etc.,
- d. Fleetmate information, including current locations and their planned paths to the next RP.

#### 2. Outputs to the GLAW loop:

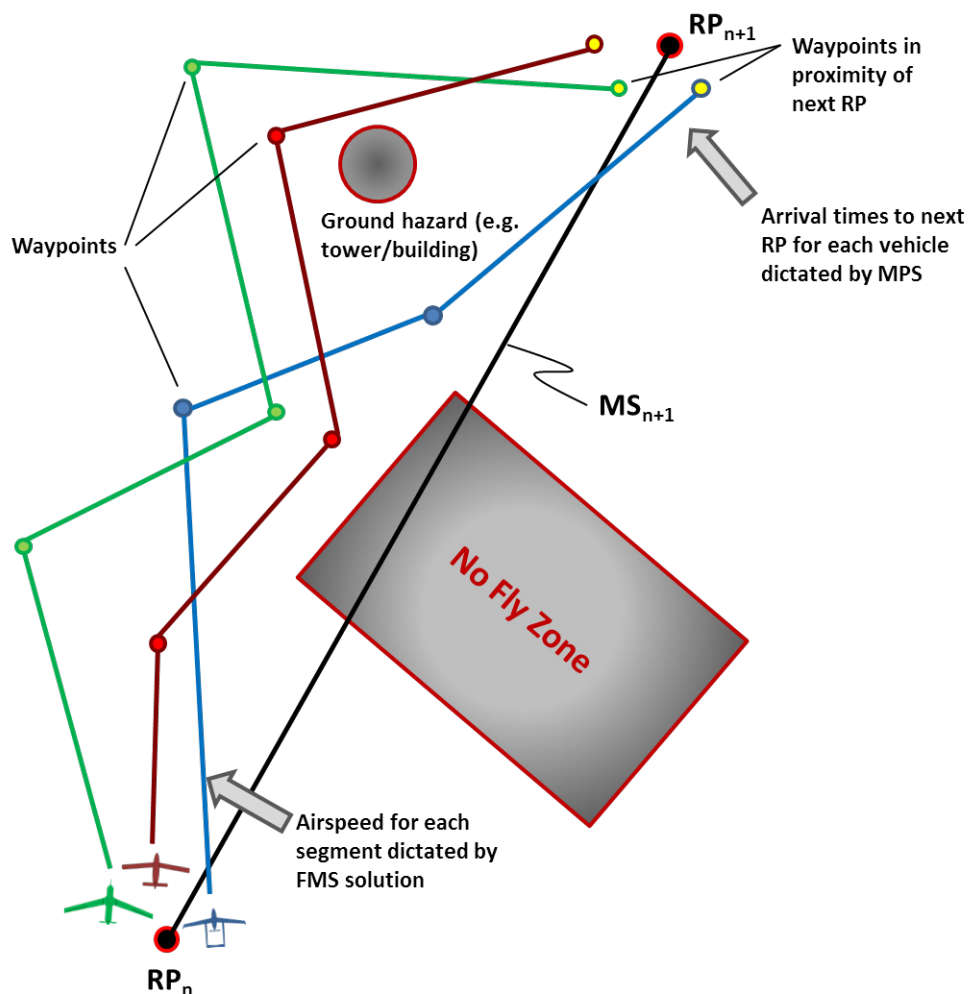
- a. An airspeed command that meets the arrival time requirements to the next RP,
- b. A series of waypoint locations in {downrange, crossrange} that progresses to a final waypoint location within the neighborhood of the next RP location,
- c. An altitude command that targets the next waypoint.

The RPs generated by the MPS define the general direction of the mission path for the fleet. These locations may also serve as *retreat and loiter* points for the fleet if anomalous conditions are encountered, required replanning or communication with ground command/control stations, for example. The FMS acts as a cooperative, distributed system. Each FMS on each vehicle in the fleet will plan out a series of waypoints that 1) avoid known obstacles or no-fly zones, and 2) avoid other fleetmates, prescribed by their current RSV distances. At each update of the FMS, all vehicles share their current waypoint plan to the next RP with all other vehicles. Through negotiation/iteration, all their paths are eventually deconflicted so that they all fly to the next RP through their series of waypoints in a safe and coordinated manner, adhering to the RSV distance requirements. This concept of operations is depicted in Figure 40 and Figure 41.

Figure 40 illustrates a mission with a fleet of three UAS vehicles (green, red and blue). The  $n^{\text{th}}$  and  $(n+1)^{\text{th}}$  RPs generated by the MPS are also depicted. Again, the MPS also delivers known hazards, which in the figure include a no-fly zone and some type of ground obstacle or hazard. The straight line segment between the two RPs may cross over a hazardous location, as shown in the figure (that is, the placements of the RPs by the MPS are not necessarily deconflicted with terrain hazards). This figure also shows a notional initial waypoint plan generated by each vehicle's FMS. Again, each FMS will plan out airspeeds for its ownship to meet the MPSs



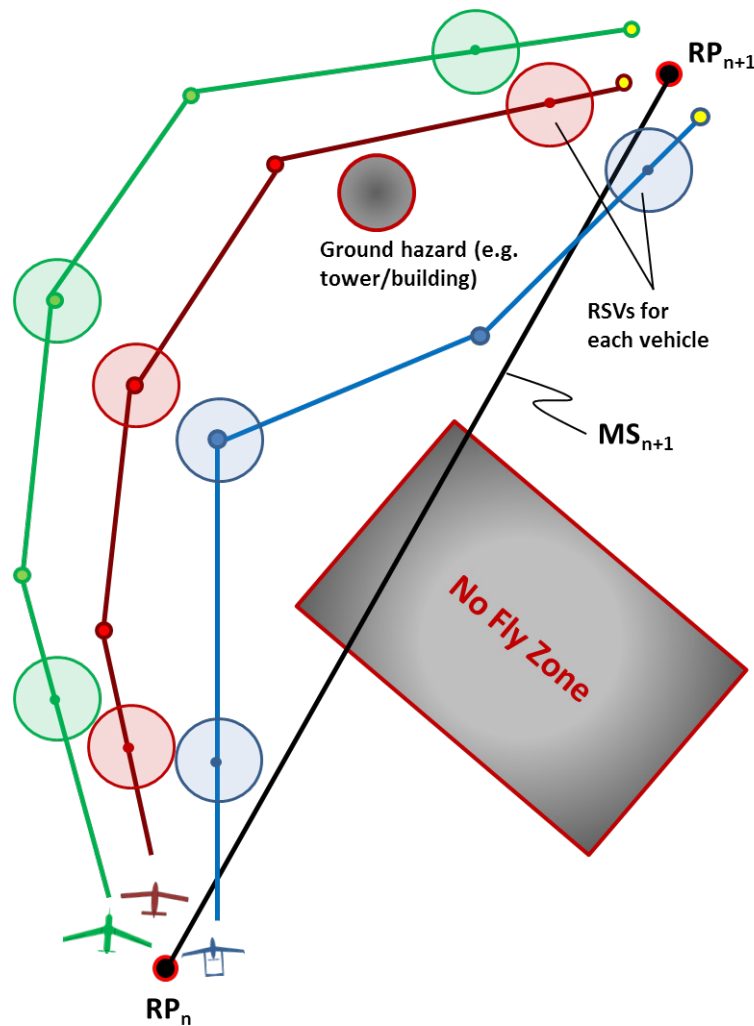
objective arrival times to the next RP. It is indicated that this initial plan has potential conflicts as the planned paths for the vehicles intersect each other at several locations.



**Figure 40. Initial Waypoint Plans Generated by FMSs**

Figure 41 illustrates the waypoint plans for each vehicle after they have all negotiated and iterated their respective plans through intra-fleet communications. There may be a number of iterations required before all conflicts are resolved. Not only are the straight line segments deconflicted, but the waypoint locations must also satisfy the RSV separation requirements, as shown in the figure. The airspeed commands for each segment may also be required to be adjusted to 1) meet the MPS arrival time requirements, 2) adhere to safety constraints (e.g., slow down through tight corridors, speed up when opportunity available), and 3) deconflict the RSVs temporally if required (e.g., again, through bottlenecks, in which each vehicle needs to align itself either in front of or behind its neighboring fleetmates). Note here that at the GLAW level, the guidance system is directly commanding the position of the physical vehicle. However, at the FMS level, the flight management function is to plan out the paths of the RSVs, both spatially and temporally. This plan must adhere to the physical limitations of the vehicle's

maneuver and airspeed capabilities. Yet, the FMS is planning out geometric cylinders through space and time, not individual vehicle locations.



**Figure 41. Deconflicted Waypoint Plans Generated by FMSs after Negotiation/Iteration**

Last, note that these figures only show the ground track for the mission. There is also an altitude component to the FMS navigation function. There may be a need to adjust the altitudes of the waypoints depending on whether there are any ground hazards that need to be cleared. Also, there may instances in which it is best to deconflict paths with neighboring fleetmates by flying at different altitudes, for example. It is for this reason that the FMS manages and generates altitude commands that are ultimately delivered to the inner-loop controller to follow through elevator control laws.

### 7.1.1 Difference between Advanced and Reversionary FMS Blocks

Again, since we are developing a modular framework, the input/output structure and basic functionality between the advanced FMS (AFMS) and the reversionary (RFMS) blocks must be the same. We consider, however, that the AFMS performs these functions using advanced, intelligent, optimal, or nondeterministic learning algorithms. We will present a candidate

advanced path planning algorithm in the subsection highlighting our experimental results. Many such algorithms are gaining wide interest because of certain optimal performance capabilities that they offer. However, these approaches would not be able to be certified with current V&V methods due to the inherent nondeterministic functions they employ.

For the RFMS, we assume here that the path planning and timing functions are accomplished using simple logic, geometric relationships, linear approximations, etc. These solutions may not always provide the optimal or best solution, but all functions and algorithms within the RFMS are considered to be capable of being certified with current accepted V&V practices. If the FMS RTA monitor commands a reversion, the RFMS has the capability to continue the mission, but simply in a sub-optimal planning manner. In fact, if a reversion at the FMS level is commanded, that does not necessarily mean the vehicle abandons the mission. We define here that the decision to abandon the mission and go to a safe or home airbase, or to continue the mission, or to perform some alternate supporting mission resides with MPS. That is, when the FMS RTA commands a reversion, the global monitor manager informs the upstream MPS of this mode switch and the FMS feedback level *waits for further instructions* from the MPS level as to what to do next.

## 7.2 Safety at the FMS Level

We consider here two aspects of safety at the FMS level:

1. Ownship safety: we focus here on two safety issues:
  - a. Existence of a safe path: does a safe path to a safe location currently exist for the ownship if required to retreat or abandon the mission?
  - b. Sufficient fuel reserves: if a safe path does exist, does the ownship currently have sufficient fuel reserves to fly to that safe location?
2. Fleet safety: does the current set of path plans keep all RSVs deconflicted, adhering to all separation distance requirements?

Again, there may be a number of other safety aspects that require monitoring by the RTA system. However, we will only focus on the safety questions stated above to construct an RTA checking framework. We therefore define the set  $S_{Dsafe}$  as the set of states such that the ownship has a safe path to a safe location with sufficient fuel reserves and that all RSVs for the fleet are deconflicted.

Having the RTA monitor continuously checking for the existence of a safe path with sufficient fuel reserves disallows the advanced FMS to from directing the vehicle into a situation from which it is unable to retreat, if needed. Again, we demonstrate the benefits of the RTA system in performing these checks in the subsection on experimental results in which an advanced path planner causes the vehicle to fly longer distances than needed. This results in depletion of fuel reserves. The RTA monitor recognizes that there is insufficient fuel to complete the mission and the MPS level commands the vehicle to abandon the mission and fly to a home airbase.

For the fleet safety question, we focused on the bottleneck problem. Fleets of UAS vehicles will typically fly at the same altitude, alongside each other due to sensor field-of-view limitations. That is, if one vehicle were to fly behind or above or below another vehicle, the other vehicle can

lose situational awareness of its neighboring fleetmate if it only has forward and side-to-side sensors. However, if a mission were to encounter a narrowing of the flight corridor due to ground obstacles or no-fly zones, then the fleet of vehicles will have to reconfigure and form a single line, for example, through the bottleneck. Once through, the fleet can then re-form their nominal positions alongside each other. The FMS RTA monitor will continually check that the required distances between RSVs are being maintained at all times. A bottleneck encounter would likely tax the FMS to adhere to this requirement and this gives us a clear example of how the RTA monitoring can provide benefit. We will revisit this problem after presenting the Type I to III safety definitions at the FMS level.

### 7.3 Type Safety at the FMS Level

The safety levels that are used to determine the switching condition protocol for an RTA protected FMS block are now formally defined. We will use the capital letter 'F' to denote that these safety definitions are for the FMS level.

**Definition 7.** FMS Type Safety - A point  $x_0$  in the state space  $S$  is:

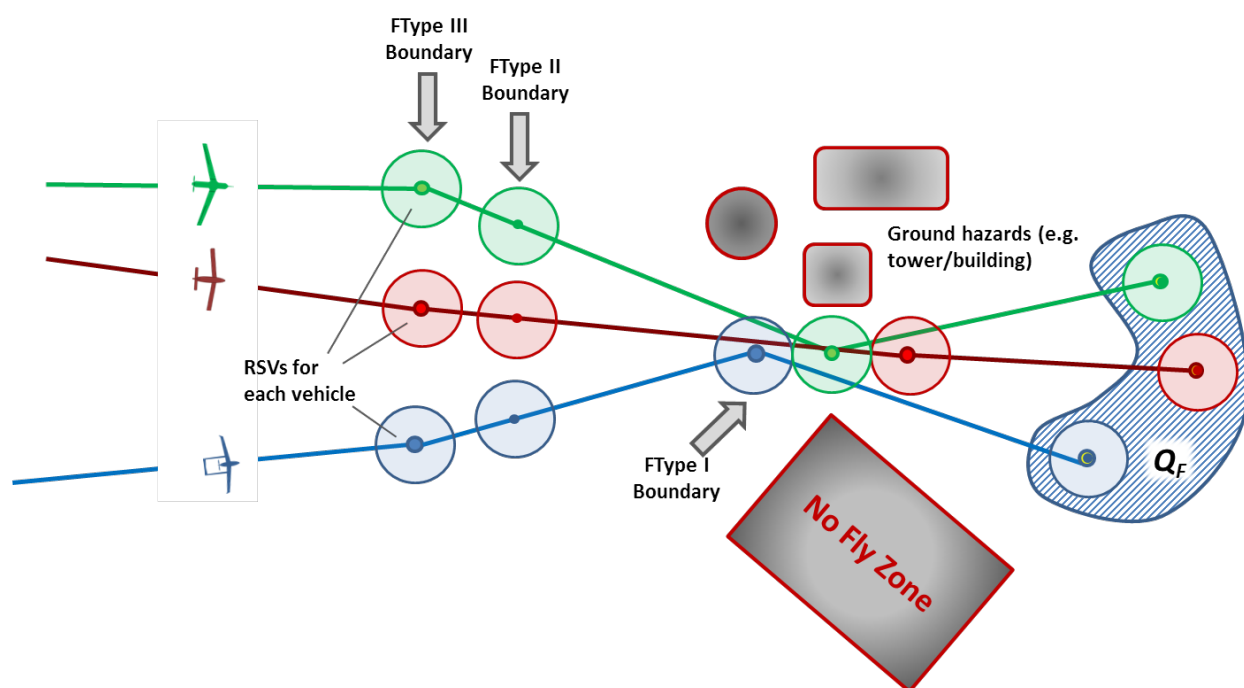
- **FType I Safe** if that point lies inside  $S_{Dsafe}$  (= existence of safe path; sufficient fuel reserves; deconflicted RSVs)
- **FType II with set  $Q_F$  and time  $T_F$  Safe** if all of the following hold:
  - a) Point  $x_0$  is FType I safe,
  - b) Upon switching to the reversionary FMS, the state trajectory can converge to at least one point in a *desired* set  $Q_F$  within a given *desired* time  $T_F > 0$ ,
  - c) The state trajectory from the point of switching to the reversionary system to the point of reaching the set  $Q_F$  is entirely contained within the FType I safe region.
- **FType III with Period  $\tau_F$  Safe** if all of the following hold:
  - a) Point  $x_0$  is FType II safe,
  - b) Every possible output of the advanced system for a time period  $\tau_F$  results in a state trajectory entirely contained within the FType II safe region.

The FMS will operate at a much lower bandwidth than the guidance and control loops. Therefore, the period  $\tau_F$  will be relatively large in value (of the order of 1 to 5 seconds, for example). The implications of this will be discussed in the next subsection.

For ownship safety, the obvious choice for the desired region  $Q_F$ , would be the nearest safe path to touchdown at a safe location, landing with sufficient fuel margin.  $T_F$  would be the time required to fly to that airbase.

For fleet safety, the bottleneck problem is illustrated in Figure 42. Again, this figure shows a fleet of three UAS vehicles next to each other flying toward a bottleneck in their flight corridor. Here, the FType I boundary is illustrated at the bottleneck and is defined as all three RSVs' boundaries just touching. If the vehicles were any closer to each other, they would be violating the required separation distances. The UAS vehicles have limits on their maneuvering capabilities and require a certain amount of time to reform into a single line while maintaining

the required separation of their RSVs. The minimum amount of time required to safely form a single line defines the FType II boundary, as indicated in the figure. If the formation of vehicles does not begin to reform by this time, then there is a risk of violating their RSVs, or worse, collision with each other or the surrounding obstacles. The FType III boundary is that last point at which the RTA monitor is assured that FType II safety will still present the next time the monitor is updated. This results in both a temporal and spatial margin between the FType II and FType III boundaries, as shown in the figure. Therefore, if the fleet has not begun to form into a single line under the management of the AFMS by the time it reaches the FType III boundary, the FMS RTA monitor will deactivate the AFMS, switch to the RFMS and inform the monitor manager of the switch.



**Figure 42. The Bottleneck Problem and FType I, II, and III Boundaries**

Once the switch to the RFMS is made, the desired region  $Q_F$  is defined as a safe state on the other side of the bottleneck, in which all three RSVs are separated with additional distance margin. Whether the fleet continues the mission with a compromised FMS level or returns to a home airbase is a decision left to the MPS level.

The actual algorithm details of how the above FType II boundary would be determined by the RTA monitor have not been explored in this program. However, we assume that such boundaries could be obtained empirically through simulation dispersion studies, or other general aircraft performance relationships involving power plant specifications, aero-braking capabilities, turning performance, etc. We assume here that such relationships would be able to be certified at design time and that enough margin would be included to assure safety robustness to uncertainties and environmental factors. One proposed requirement would be that if a bottleneck is determined to exist within the flight corridor, then the fleet should begin forming a single line at a specified distance from the bottleneck. That specified distance would be defined

based on design time simulation studies and other analyses. Risk considerations could also be included. For a mission that is able to accept more risk, the specified distance can be allowed to be at its minimum, for example, whereas for a more conservative mission involving high value assets, the specified distance will be increased in length to provide the fleet with more time margin to safely maneuver through the bottleneck.

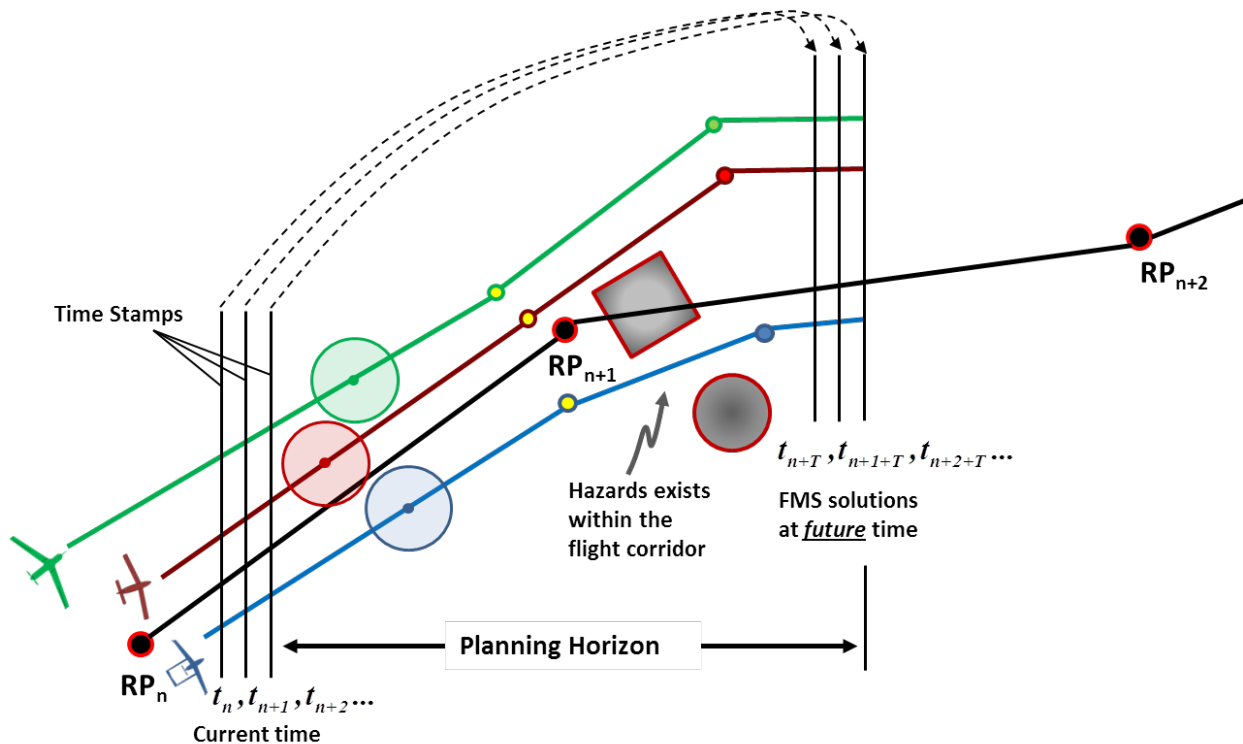
### 7.3.1 FMS Planning Horizon

Although the guidance and control loops must operate in real time, the FMS and MPS levels will operate in *future* time. That is, during operations, the MPS will be planning out the entire mission, *globally*, as well as over a relatively long time horizon (say, 1 to 5 minutes), *locally*. For example, it may be planning out and delivering three or four RPs ahead of the current time as its current local solution. As more information comes into the MPS, it will continually adjust the timing and RP locations, as warranted. The FMS will then operate over a shorter time horizon, (say, 0.5 to 1 minute); however, this time horizon is still large compared to real time operations experienced at the guidance and control levels. The reason both the MPS and FMS levels should be planning out the mission and pathways in the future (on multiple time horizons) is to account for unexpected obstacles or unforeseen events and *plan around* these hazards with sufficient time margin to ensure that a safe course of action is ready and available once the fleet arrives at these identified hazards and threats. Certainly there will be the potential for immediate, emergency scenarios requiring evasive action. This topic will be discussed in Chapter 8. However, if the fleet can plan into the future, it is prudent to do so, given whatever information and intelligence it has at the current time. An analogy here is a commercially available GPS route planning device or smart phone application. If it has current information regarding traffic congestion, it may warn the driver and suggest an alternate route – *long before the driver has entered into the traffic*.

This future planning protocol is illustrated in Figure 43. Here, the fleet is currently flying between  $RP_n$  and  $RP_{n+1}$ . Since the FMS plans out future waypoint locations, the paths for each vehicle have already been planned out and deconflicted for this mission segment. While the fleet flies between these two RPs, the FMS is already planning out waypoint locations and deconflicting paths for the mission segment between  $RP_{n+1}$  and  $RP_{n+2}$ , taking into account all known obstacles and hazards. Notionally, the planning horizon should extend beyond the next RP, although the horizon may be either fixed or vary in length, depending on the mission scenario and other considerations. As depicted in the figure, however, by the time the fleet arrives at  $RP_{n+1}$ , the FMS should already be planning out paths beyond  $RP_{n+2}$ .

The implications of this are as follows. The previous scenario regarding the bottleneck problem will not occur in real time, but rather in future time. That is, the FMS RTA monitor will be continually checking for loss of FType III safety *on the future solutions* that are generated by the AFMS. The FMS RTA monitor will not wait until the fleet has actually arrived at the FType III boundary before switching to the RFMS. It will switch to the RFMS as soon as it has determined that loss of FType III safety has occurred for the future AFMS solution (at the planning horizon time). The reason for this is that if loss of FType III safety occurs, then this indicates that the AFMS is broadcasting out an incorrect solution and can no longer be trusted. Consider, for example, that the FMS block is updated at an N Hz frame rate, say every 10 seconds. Then, within than time all the vehicles in the fleet will negotiate, iterate and ultimately deconflict their paths. At the end of the frame, all the AFMS instantiations within the fleet

should have an agreed upon solution, which is then broadcast to all the FMS RTA monitors. By the act of broadcasting its solution, the AFMS is *confirming* that it believes this to be a correct solution. If the RTA monitor disagrees with that assessment, then the AFMS can no longer be trusted to provide correct solutions and it should be disengaged and its function switched to the RFMS. In this manner, since there is no immediate threat of loss of safety, *the FMS RTA can be defined as a software integrity monitor rather than a safety monitor.*

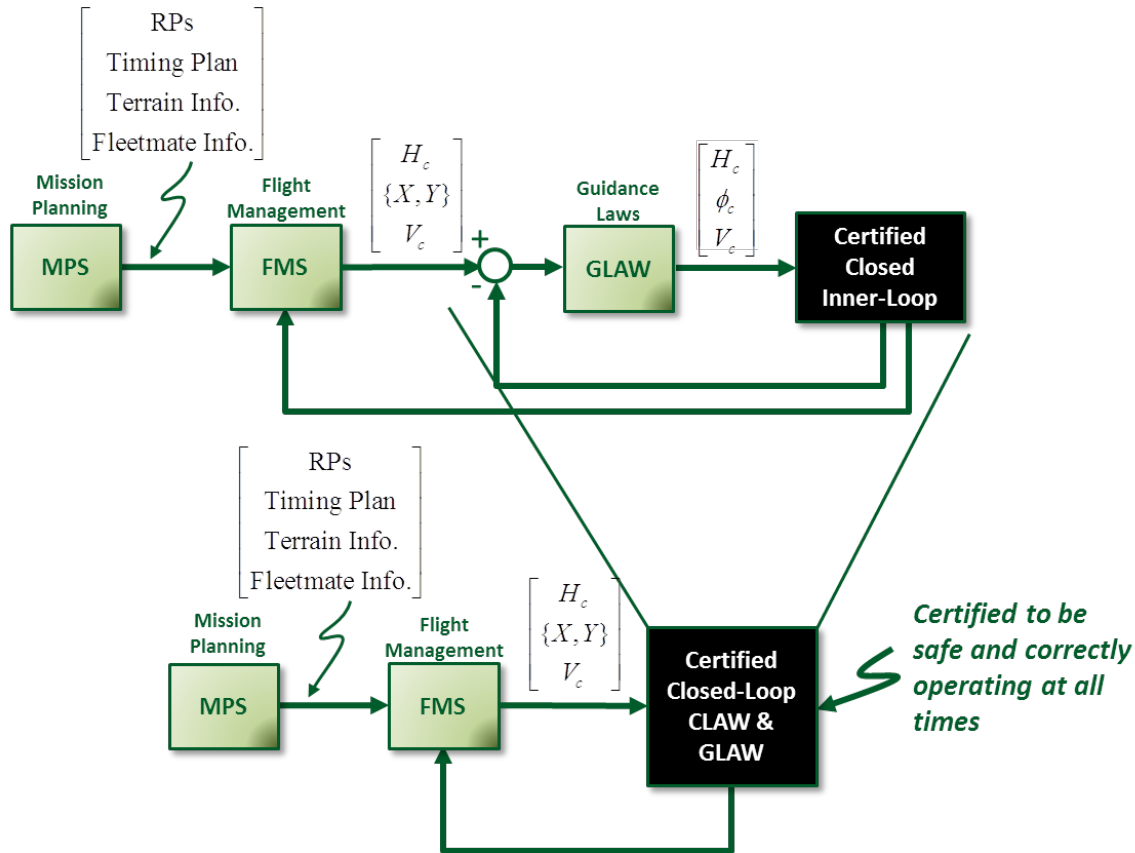


**Figure 43. FMS Plans Out Path at Future Time**

Notionally, with a long planning horizon (again, 0.5 to 1 minute, for example), there is substantial time between when the FMS RTA has determined the AFMS is not delivering a correct solution and when actual unsafe conditions will occur. One may argue that before shutting down the AFMS, it should be allowed to correct itself and provide a new solution since there is time for such corrective actions to be taken. We do not recommend this protocol because 1) trust or integrity is lost at the first occurrence of an incorrect solution, and 2) such iteration between the AFMS and the FMS RTA would most likely require specific design integration between the two systems, contrary to the modular, plug and play framework we are recommending.

#### 7.4 A-G Contracts at FMS Level

Just as with Figure 23 and Figure 36, Figure 44 illustrates how the outer-loop GLAW feedback level is collapsed into the composite *black box* that is certified to be safe and correctly operating at all times (by the compositional reasoning logic). We now develop the A-G contracts at the FMS level for the elements around the bottom feedback loop shown in this figure.



**Figure 44. Development of A-G Contracts at the FMS Level**

#### 7.4.1 Assumptions on Inputs from the Upstream MPS

The following assumption on the inputs to the FMS loop can be listed:

- i. Communication hardware is working correctly (radios for intra-fleet transmissions working)
- ii. Sensors to detect hazards/obstacles working properly
- iii. Fleetmates are operating correctly; their FMS instantiations are correctly negotiating and executing solutions; fleetmates are correctly communicating their solutions in a timely manner and adhering to the frame rate of the FMS block.
- iv. Ownship and fleetmate vehicles are physically capable of arriving at designated RP locations, at or near to the commanded arrival times
- v. Hazard and other theater/environment information is accurate to within specified tolerances.

In summary, all fleet hardware is working properly and the mission plan is physically achievable. We have not addressed cases in which a fleetmate vehicle is not working properly or incapable of communicating with the rest of the fleet, due to equipment malfunction or battle damage, for example. We assume here that mitigation procedures would be in place for such cases and the *broken* vehicle would leave the proximity of the fleet, adhering to proper protocols.

#### 7.4.2 A-G Contracts for Closed-Loop CLAW/GLAW System

The assumptions on the inputs to the closed-loop guidance and control system are:



- i. Altitude and airspeed commands and their rates lie within acceptable upper and lower bounds (see Eq. (1))
- ii. Positioning of waypoints shall not require climb, descent or turning performance beyond vehicle's capabilities
- iii. Waypoints shall not be spaced so close together that vehicle cannot properly follow them
- iv. Waypoints shall not be placed in hazardous locations (near objects, terrain that can lead to collisions, etc.)
- v. Waypoints shall not be placed in restricted airspace or no-fly zones.

The guarantees of the closed-loop guidance and control system are:

- i. Vehicle is stable in attitude
- ii. Vehicle is stable in translation
- iii. Vehicle is accurately following commanded path (waypoints), remaining within its defined RSV or safety corridor.
- iv. Vehicle is accurately following commanded airspeeds
- v. Stability and path following performance robust to acceptable disturbances and levels of uncertainty.

In summary, the vehicle is performing correctly and accurately flying to the commanded waypoint locations.

#### **7.4.3 A-G Contracts for the FMS**

The assumptions on the inputs were presented in Subsection 7.4.1. The guarantees of the FMS are the following (waypoint locations refer to the 3-dimensional location in space in some coordinate frame: {downrange, crossrange, altitude}):

- i. A safe path to a safe location is currently available to the ownship
- ii. Ownship has enough fuel reserves to get to the safe location by that safe path
- iii. Airspeed commands generated by the FMS are such that the ownship can stay within its safety corridor or RSV
- iv. Waypoint locations generated by the FMS avoid all known obstacles, no-fly zones, ground, and fleetmates
- v. Waypoint locations are positioned such that ownship's defined safety corridor or RSV is deconflicted with all known obstacles and fleetmates' safety corridors or RSVs
- vi. Path is properly deconflicted with fleetmates through negotiations
- vii. Intra-fleet communications correctly pass path plans to/from each vehicle in the fleet
- viii. Solution (waypoint locations, commanded airspeeds) for next MS completed before arriving at next RP
- ix. Waypoint location adhere to defined protocols (e.g., TERPs) such that all vehicles in the fleet can physically and safely follow the waypoints (path defined by waypoints is flyable)
- x. Commanded airspeeds are such that the MPS timing plan is achieved (ownship and fleetmates arrive at next RP at the proper designated times (within some acceptable margins)).

Although we have not explored performance measures for the FMS, there may be some defined performance metrics that could be included as guarantees (or design requirements). These may

involve some objective function value on fuel usage or measures of progress to the next RP, or safety margins being maximized, for example.

#### **7.4.4 A-G Contract for FMS RTA System**

For the reversionary FMS, the A-G contracts will be equivalent in structure to the FMS in general. However, specifically, the reversionary FMS also guarantees that at least FType I Safety holds for all time (a safe path exists for all time with sufficient fuel reserves and all RSVs or safety corridors are deconflicted for all time). Additionally, the desired region  $\mathcal{Q}_F$  is achievable within time  $T_F$  when the RFMS is first activated by the RTA monitor and switch mechanism. Again, however, this last guarantee may not be necessary since the switch to the RFMS will occur long before safety becomes an issue.

For the RTA monitor and switch mechanism block, the main assumptions are that the information input to this block is valid (correct to within some accuracy tolerance) and that it at least starts its operation while the system is within the FType III safety envelope.

As long as the above assumptions hold, the guarantees for the RTA monitor and switch mechanism are that:

- i. Loss of FType III safety is always detected at first occurrence, while the system is in the FType II safe region, and
- ii. The reversionary FMS is activated when loss of FType III safety is detected.

Again, since the above will occur in future time, the guarantees are essentially that the FMS RTA monitor will always detect an incorrect planning solution and switch to the RFMS once this occurs. The last guarantee is that the FMS RTA will always communicate the reversion switch to the upstream MPS.

### **7.5 FMS RTA Checks and Switching Conditions**

Recall, the general checks the RTA monitor must perform were listed in Subsection 3.4.1. We can now specify these checks for the FMS level.

1. Safety check: At each update to the FMS RTA monitor, it checks that:
  - a) The ownship has a safe path to a safe location,
  - b) The ownship has sufficient fuel reserves to achieve that safe location,
  - c) The future system state to determine if it has left the FType III safe region. If so, the advanced FMS is shut down and function is switched to the reversionary FMS.
2. Output/environment check: The FMS RTA monitor checks that the airspeed and altitude commands and the waypoint locations do not violate any constraints imposed on the closed-loop guidance and control system. Again, this will involve rate and magnitude limits on the commands and other constraints on the waypoint locations, as previously listed.
3. Performance check: The RTA monitor checks that the advanced FMS is achieving its minimum required performance, however defined and measured.

4. Input/environment check: The RTA monitor also checks that inputs to the FMS level do not violate any constraints imposed on the inputs. Such constraints will include proper placement of RPs and achievable arrival times to those RPs.
5. System hardware health check: As with the other loops, assumptions on hardware and information integrity should be checked and mitigation strategies coordinated with an IVHM or RM system.

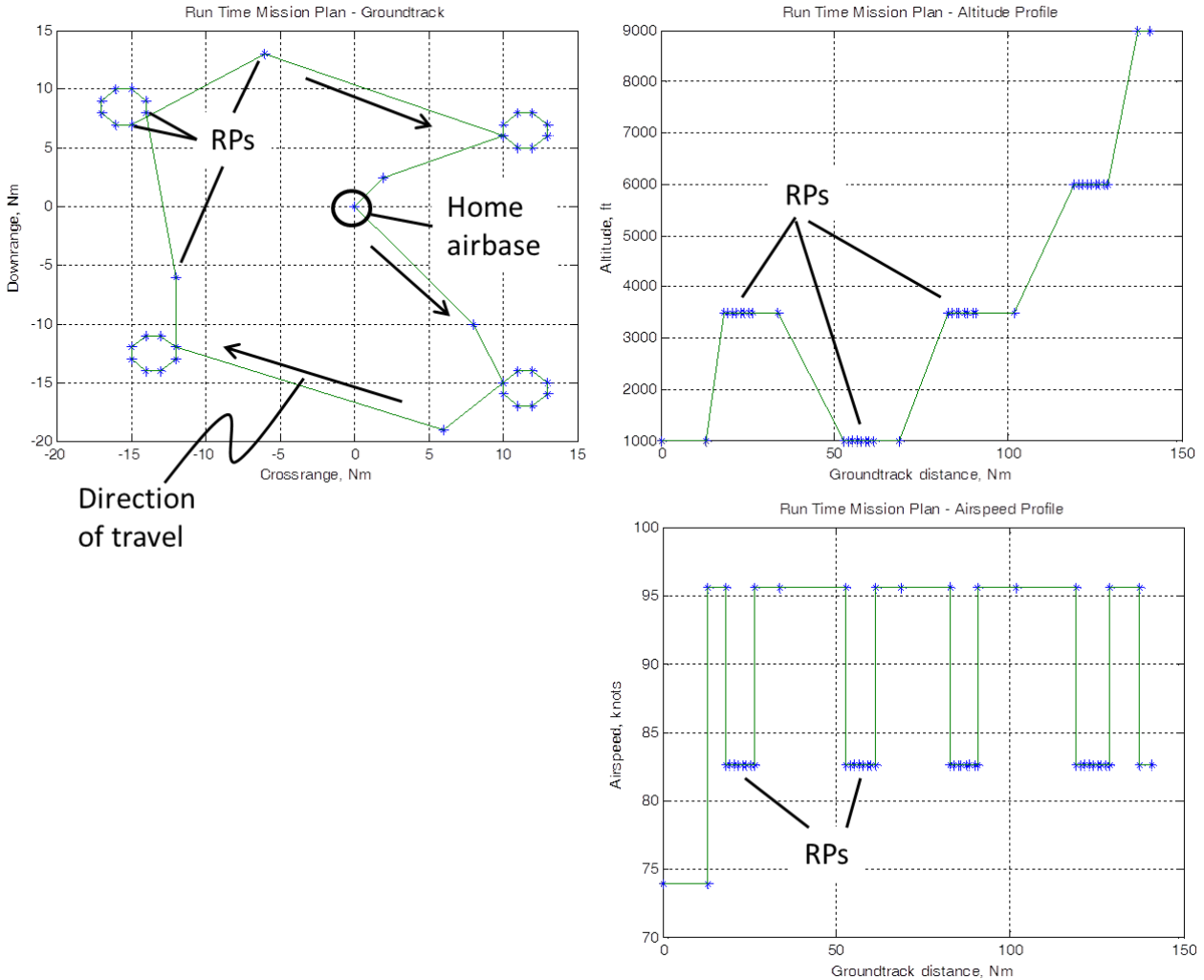
## **7.6 Experimental Results**

### **7.6.1 FMS RTA Check on Safe Path Existence and Fuel Reserves**

We used our developed Matlab/Simulink model of the 3-DOF UASs to perform experiments on the FMS RTA checking for existence of a safe path to the home airbase and if sufficient fuel reserves are present to fly to the airbase. Figure 45 presents a notional mission plan generated by the MPS that is used in the current case study. This figure presents a ground track (downrange vs. crossrange), an altitude profile (ground track vs. altitude) and an airspeed profile (ground track vs. airspeed). For simplicity, we assigned airspeeds for each MS (we did not code up any algorithm that would convert a timing plan to an airspeed profile). The RP locations are plotted as blue ‘\*’ and it is seen that the mission leaves a home airbase and flies in a clockwise direction around to four different circles defined by RPs. These could be, for example, installations or populated areas to be surveilled. The area covered is approximately 45 by 35 Nm and the altitudes range from 1,000 to 9,000 ft. The airspeeds range from approximately 75 kts to 95 kts. These parameters would be typical of a mission for Tier I or II UASs executing a surveillance mission.

Now, consider the situation that arises in Figure 46. In this case, there is a defined no-fly zone outlined in red with the right end impinging on the straight-line connection between rendezvous points that requires the FMS to plan a path around the no-fly zone. This no-fly zone could be a mountain range or enemy territory and extends for approximately 50 Nm to the left in the figure.

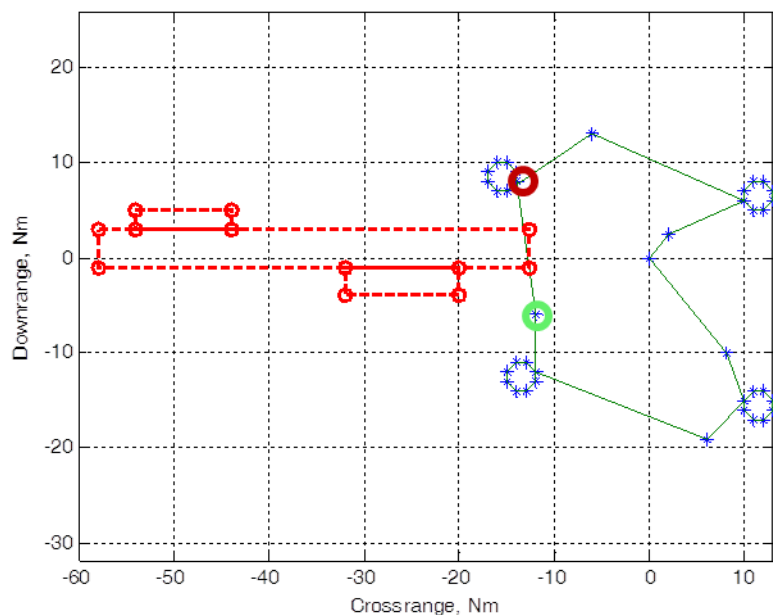
In this example, we investigated an advanced path planner for the AFMS known as the RRT\* method that generates the waypoints to avoid this no-fly zone. A number of methods have been used to solve path-planning problems including graphed-based approaches such as Dijkstra's Algorithm, A\*, LRTA\*, D\*; Monte Carlo or sample-based techniques such as probabilistic roadmaps (PRM) [Kavraki 1996] and Rapidly-Exploring Random Trees (RRT) [LaValle 2001]. Sampling-based motion planning algorithms offer the potential to find feasible solutions to problems with high-dimensionality while retaining the dynamics and constraints of the system under analysis. Recent extensions have shown the ability for certain sampling-based motion planners to operate in real-time. Sampling-based motion planners randomly select states in a specified state-space and check for connectivity of the sampled state with existing states. Each new connection expands a set of connected states and this propagation continues until a complete path from the start and goal states is found.



**Figure 45. Rendezvous Points and Airspeed Schedule for Case Study**

A lack of optimality is the primary drawback of the RRT family. While the probability of finding a solution approaches one as the number of states increase, the path is not guaranteed to be optimal. The RRT\* algorithm was developed to address this limitation. Primarily developed by researchers at MIT [Karaman 2013,2011,2010,2008,2011(a),2013(a)], [Kuwata 2009], [Luders 2010], [Joel 2011], RRT\* provides asymptotic optimality of the generated path. The original RRT algorithm, while advantageous to find paths quickly in high-dimensional spaces, may return a path that is far from optimal. RRT\* basically operates by first searching for an initial feasible path and then refining and optimizing that path during the remainder of the time to traverse the path and reach the goal state.

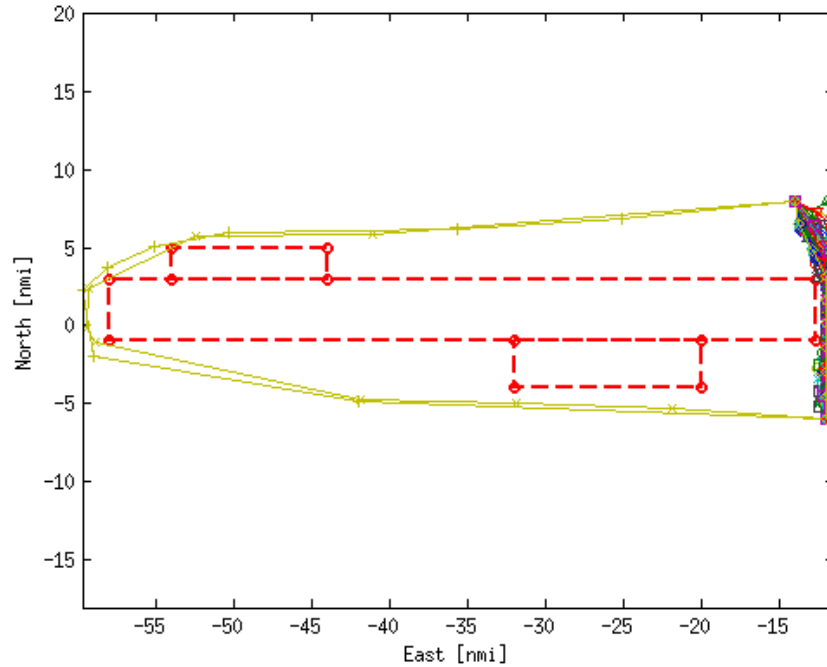
Prior to path planning, the AFMS accepts rendezvous points from the MPS and assigns appropriate waypoints to the downstream guidance and control loops to follow. In this case study, there is only one vehicle, and for simplicity, we equate the rendezvous points as the waypoints to follow. In Figure 46, the green circle indicates the starting waypoint and the red circle is the end waypoint. Between these two waypoints we exercised the RRT\* path planning algorithm.



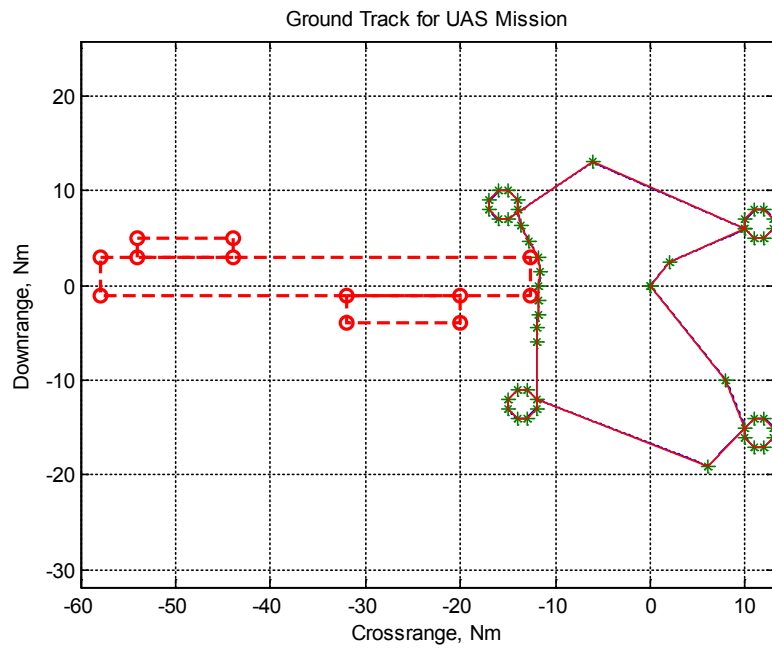
### Figure 46. No-Fly Zone Interference with Nominal Mission Path

Due to its stochastic nature, the RRT\* algorithm implemented in the AFMS will not always return the same path for a given set of inputs. For example, Figure 47 presents 103 paths returned (one per run) by the RRT\* algorithm given the beginning and ending waypoints shown in Figure 46. Most (101) of the returned paths avoid the obstacle to the right; these paths are visible to the right of the obstacle in Figure 47. Two of the runs, however, returned the much longer paths that are depicted around the left side of the obstacle in Figure 47. These are clearly non-optimal solutions and the reason they resulted was due to the time limitations from the update rate of the FMS block. Given more time, these solutions would have been corrected. The implication here is that during real time operation, effects of computational limitations can cause erroneous solutions of an otherwise good algorithm. In most cases, the vehicle will follow the short path and avoid the obstacle to the right. However, the possibility exists that occasionally (2 out of 103 occurrences in this example) the vehicle will follow the much longer path to the left. In actual implementation of such an advanced algorithm this rate of occurrence would not be acceptable, but even if it were improved to a 1 in 1,000 or 1 in 10,000 chance, there would still be the need for an RTA system to provide protection since there is no way to predict the behavior of such a nondeterministic planner at design time.

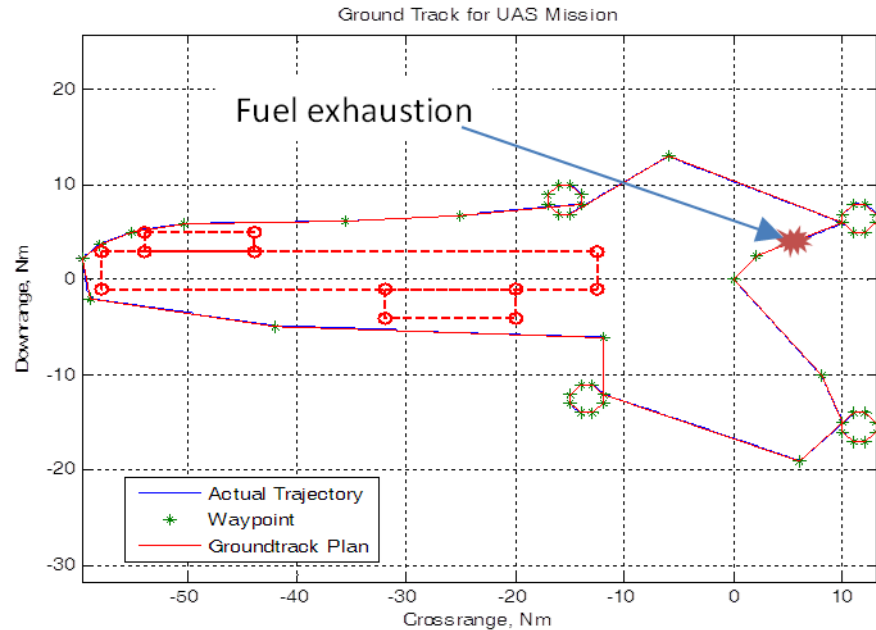
Figure 48 presents the successful completion of the mission when the RRT\* algorithm within the AFMS module returns the short avoidance path to the right of the no-fly zone. In contrast to this, Figure 49 presents the result if the AFMS follows the long avoidance path to the left. In this case, this results in the vehicle running out of fuel and is unable to make it back to base. The plot in Figure 50 presents the fuel quantity (in slugs) remaining for the case when the long RRT\* path is flown.



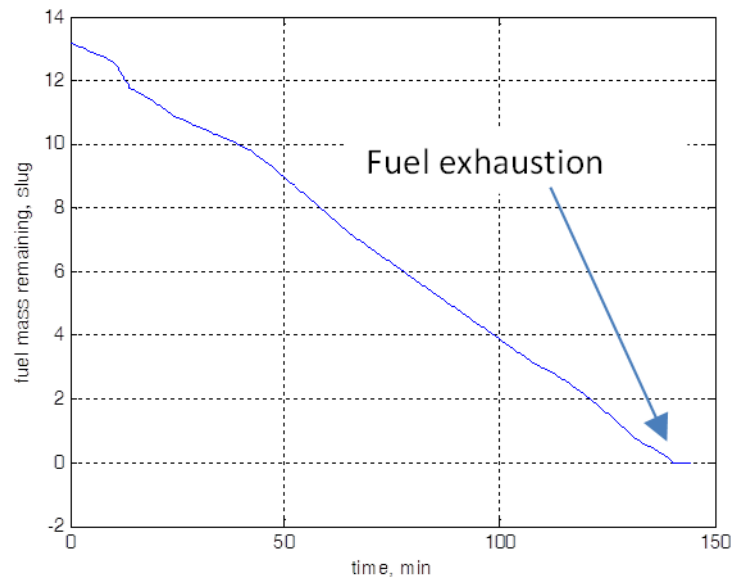
**Figure 47. 103 Paths Returned by RRT\* for the Same Pair of Advanced Waypoints**



**Figure 48. Case Study Following Short RRT\* Computed Path - Mission Completed**



**Figure 49. Case Study Following Long RRT\* Computed Path – Aircraft Runs Out of Fuel**

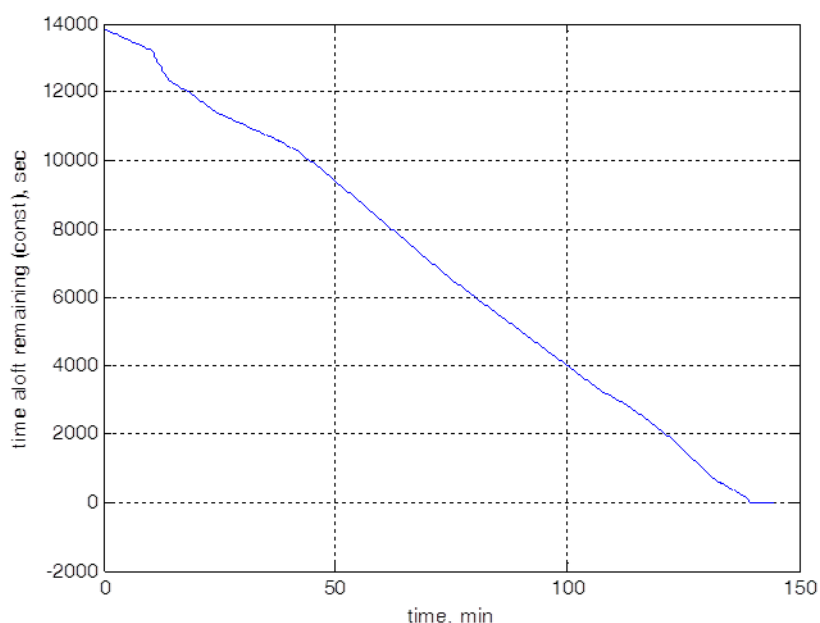


**Figure 50. Fuel Remaining during Mission with Long RRT\* Avoidance Path**

The above example motivated the construction of the FMS RTA check on fuel reserves. More generally, we modeled the RTA checking *endurance reserve*. Endurance reserve is an estimate of the available flight time, based on remaining fuel, that would be left over when arriving at base if the decision to return to base were made at the current time. (e.g., if a vehicle currently has a 30-minute reserve endurance, that implies if the vehicle return to base at the current time, it will land with enough fuel left over to fly for 30 more minutes.

Computing the estimated reserve endurance is based on the difference between the time-aloft-remaining estimate and the time-back-to-base estimate. The time-aloft-remaining is an estimate of the time remaining, at a given throttle setting, until fuel exhaustion. This is based on the expected fuel burn rate at the given throttle setting and the quantity of fuel remaining. The time-back-to-base is an estimate of the time required, at a given throttle setting, to get back to base. This is based on the airspeed at the given throttle setting and the distance back to base. This estimate also requires a winds-aloft forecast or a worst-case expected winds estimate. The impact of winds will be discussed later.

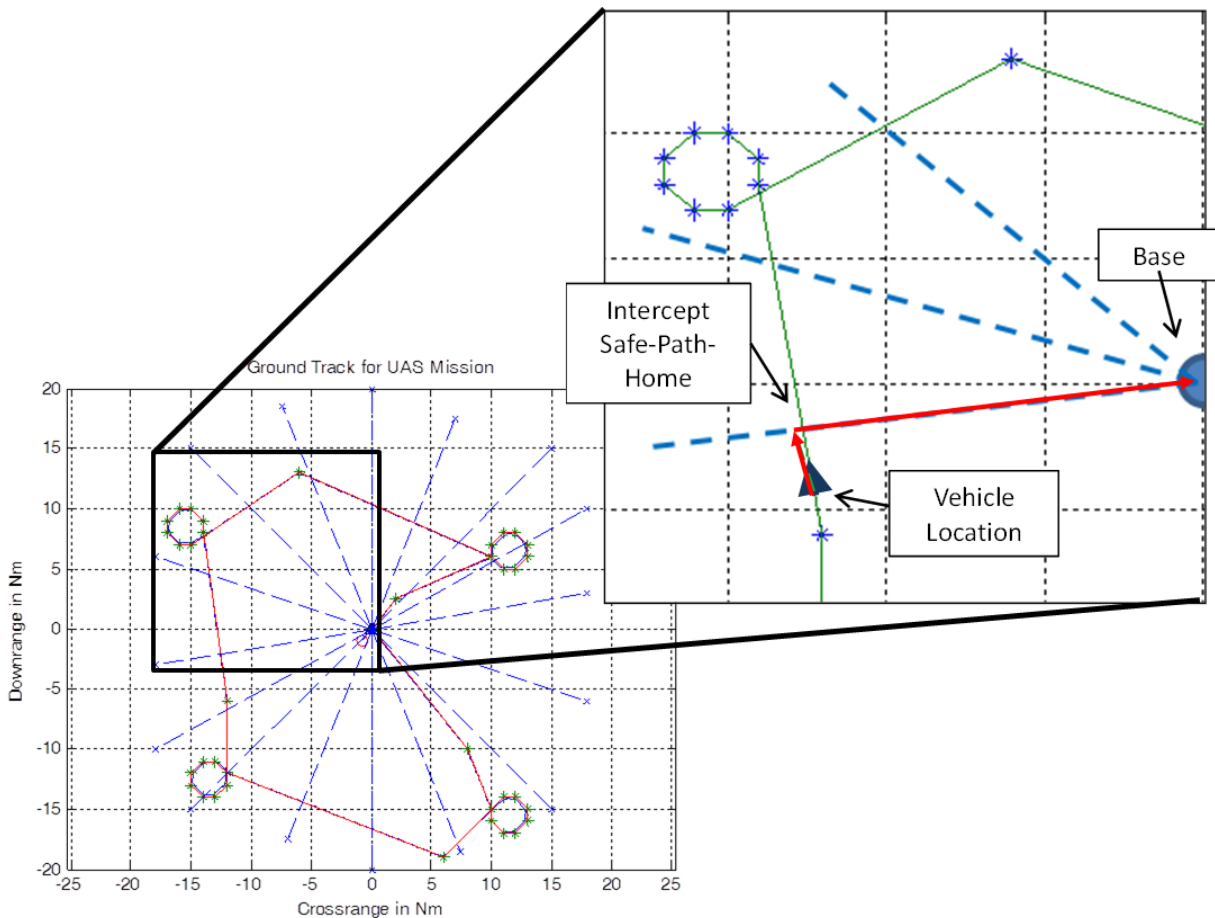
To illustrate the notion of reserve endurance, Figure 51 presents the time aloft remaining for the long avoidance case presented above. Whereas, Figure 50 presented the actual fuel mass remaining, Figure 51 presents an estimate of the flight time remaining by multiplying the mass remaining by an expected flight time per unit of fuel mass. The flight time expected is 1085 seconds per slug at a throttle setting that corresponds to 29.5 lbf of thrust and 125 ft/s airspeed.



**Figure 51. Time-Aloft-Remaining**

Estimating the time required to get home requires an estimate of the distance to base. The MPS level continuously provides the FMS level terrain and theater data and we assume here that the FMS either calculates onboard or queries a database certified safe paths home. These safe paths home are represented by the dashed blue lines in Figure 52. To estimate the distance back to base, the FMS RTA calculates the perpendicular distance from the current location to the nearby safe-paths-home then adds the calculated perpendicular distance to the distance along the safe-path-home from the perpendicular intercept to the base. The sum of these two legs is the total distance home. The close-up in Figure 52 presents an example for illustration. The distance home from the current vehicle location is represented by the solid red line with two legs.

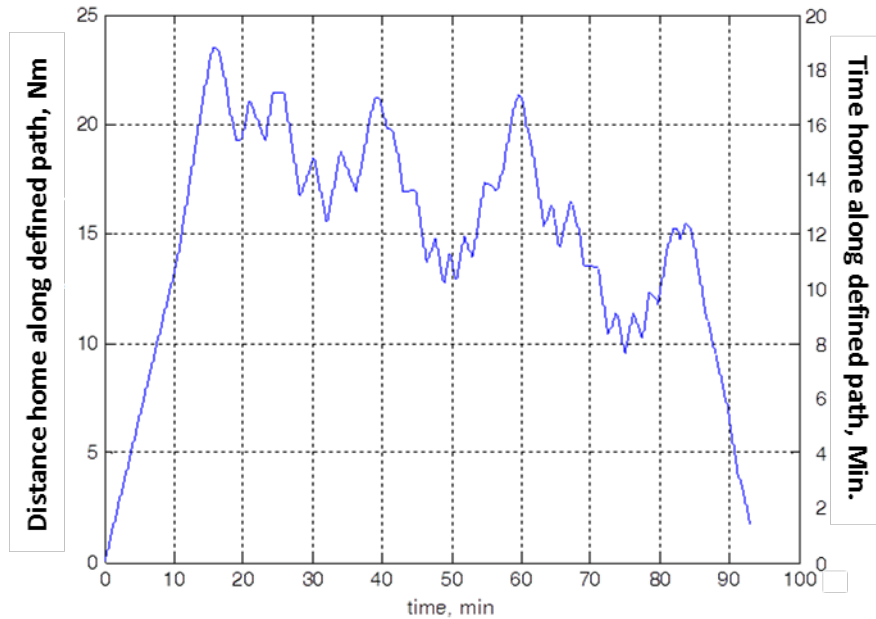




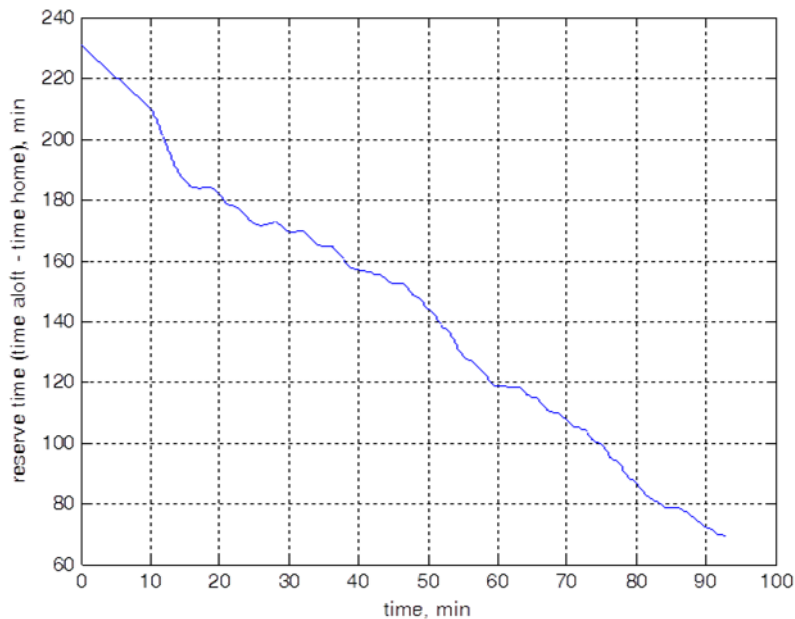
**Figure 52. Sample Path Home from Current Location**

During the mission, the shortest distance home from the current position is continuously computed by the FMS RTA using the set of known safe paths home and the perpendicular distance to those paths. Figure 53 presents the resulting shortest distance home in Nm (left-hand axis) as the vehicle executes the mission. Now that the distance home is known, the last step in determining the time-to-get-home quantity is estimating the time it will take to follow the computed shortest path home. The time to follow a path home depends heavily on the wind speeds aloft. In the current example the winds aloft are assumed to be zero. The impact of winds will be addressed later. The resulting shortest *time* home based the shortest *distance* home, together with the assumption of zero winds aloft and a true airspeed of 125 ft/s results in the same plot shown in Figure 53, but expressed in minutes (right hand axis). This 125 ft/s is the resulting airspeed consistent with the throttle setting used to determine the time aloft remaining.

Finally, with the shortest-time-home and with the remaining-time-aloft, the reserve endurance estimate can be determined. The reserve endurance presented in Figure 54 is found by subtracting the shortest-time-home (Figure 53) from the time aloft remaining (Figure 51). If, as an example, the decision to go home were to be made 75 minutes into the mission (the x-axis of Figure 54) then operators would expect approximately 100 minutes worth of fuel remaining in the fuel tank (y-axis Figure 54) of when arriving at back at base.



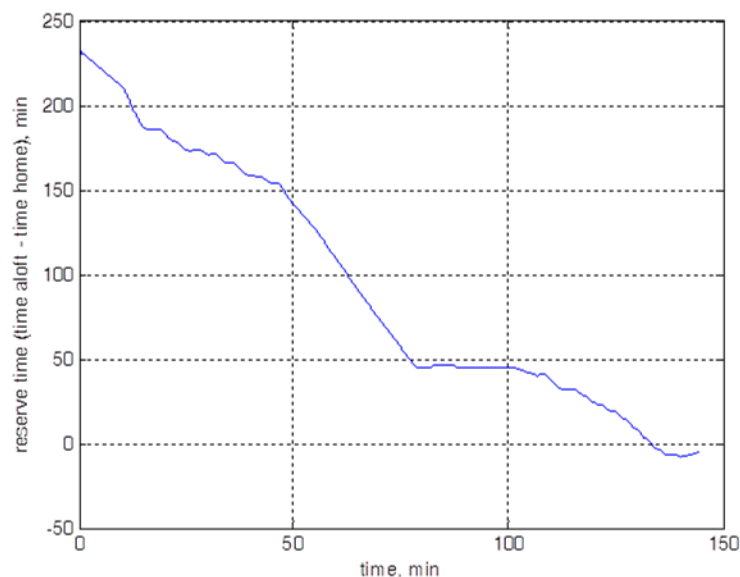
**Figure 53. Shortest Distance and Shortest Time Home along Defined Path**



**Figure 54. Estimated Reserve Endurance**

To use estimated reserve endurance as an RTA safety check, the designer must simply specify a desired minimum acceptable reserve endurance threshold. Then, if the reserve endurance drops below that threshold, the RTA will switch to the reversionary mode to guide the vehicle home along a pre-specified safe path home. The expectation is that, when arriving back at base, the vehicle will have the specified endurance worth of fuel remaining on board. The next step is to implement this safety check on the above mission that resulted in fuel exhaustion. The RTA check is expected to trigger a reversion and guide the vehicle back to base safely before it runs out of fuel.

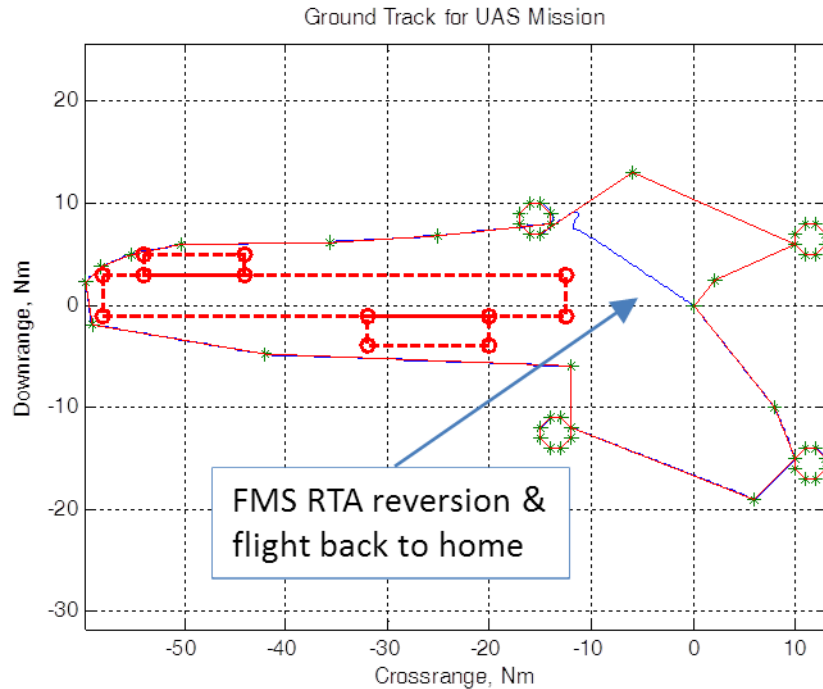
Figure 55 presents the estimated reserve endurance associated with the failed mission presented in Figure 49. Reserve endurance below 0 means that, although the fuel may not have run out yet, there is not enough to get back to base. For the current case study, the reserve threshold is selected to be 30 minutes meaning that, when the reserve gets down to 30 minutes, the aircraft should fly back to base and have approximately 30 minutes remaining after landing. This notion is closely related to *bingo fuel* which is the fuel state at which a mission is aborted in order to arrive back at base with desired reserve fuel. Also, selecting a reserve endurance threshold above 0 is similar to traditional planning and decision making which is done conservatively to account for uncertainty (wind, vehicle variation, etc.).



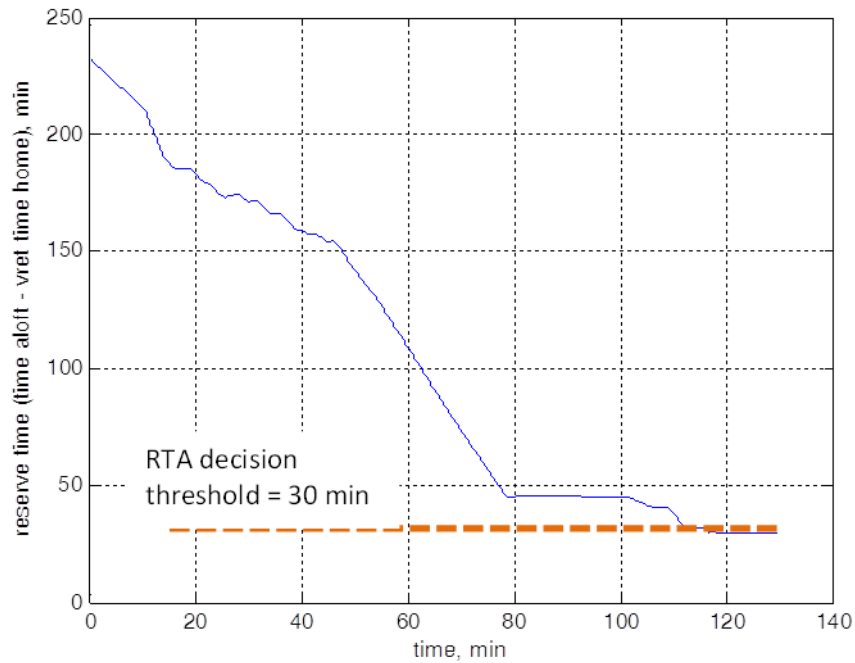
**Figure 55. Long RRT\* Path Case Study - Estimated Reserve with No RTA Protection**

Figure 56 presents the protection afforded by the use of reserve endurance RTA. In this case, the RTA protection triggers a reversion and the RFMS guides the vehicle safely back to base before its fuel is exhausted. Figure 57 presents the associated reserve endurance estimate together with the RTA threshold of 30 minutes reserve. As shown, once the reserve endurance is depleted down to the 30 minute threshold, the RFMS guides the vehicle home and the vehicle arrives back at base with 30 minutes of fuel on board.

Recall that there is zero wind in this case study. In practice, there is always wind as well as myriad other sources of uncertainty including variations in vehicle characteristics (engine efficiency, airframe condition, etc.) as well as environmental characteristics (temperature, density, etc.). The reserve threshold is chosen conservatively to accommodate these uncertainties. However, if the threshold is chosen too conservatively, the RTA will trigger a return-to-base reversion early thereby prematurely terminating the mission. The chosen threshold is therefore a tradeoff between the importance of completing the mission and the risk of losing the vehicle.



**Figure 56. Long RRT\* Path Case Study – RTA Reversion Back to Base**



**Figure 57. Long RRT\* Path Case Study – Reserve Endurance and Threshold**

We therefore next investigated the impact of winds as well as the impact of errors in wind forecasts. The key difference between the no-wind results presented previously and RTA decision making in the presence of winds is that the wind speed and direction must now be taken into account when computing the time-required-to-get-home quantity. Headwind components of the wind vector slow the vehicle groundspeed while tailwinds increase the groundspeed. As

such, a forecast of winds aloft (speed and direction) is useful for calculating the time-to-get-home-quantity. Alternatively, a worst-case winds estimate may be used but this option may lead to overly conservative RTA decision making.

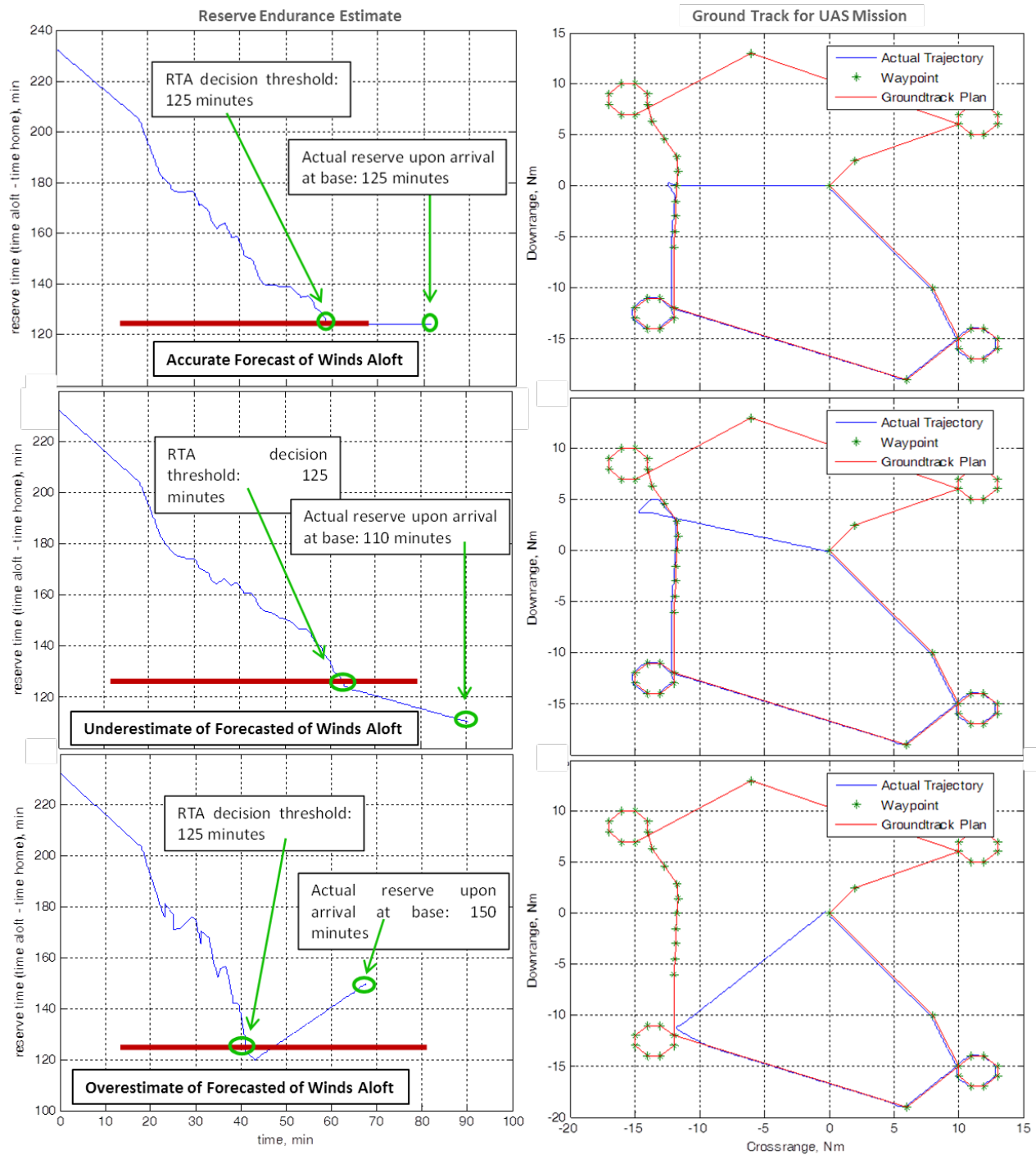
We include three example cases to highlight the outcome with 1) accurately predicted winds, 2) an underestimate of the adverse effect of winds, and 3) an overestimate of the adverse effect of winds. As expected, an underestimate leads to a late return-to-base decision and less reserve upon arrival than desired while, conversely, an overestimate leads to a premature return-to-base decision and an excess of reserve upon arrival at base. These experimental results are presented in Figure 58. The plots in the first column present the reserve endurance, and the plots in the second column present the resulting flight back to base once the RTA decision is triggered to revert to the RFMS. For all case studies, a RTA decision threshold of 125 minutes of reserve endurance is defined under the presence of an actual 40 knot steady wind from the East.

The first case presents the results for accurate forecast of winds. The reserve endurance reaches the 125 minute threshold just before 60 minutes (mission time) and, because the accurate wind forecast allows for accurate planning, 125 minutes of reserve fuel remains when the vehicle arrives back at base.

The next case shows an underestimated forecast of wind speed (35 knots) provided to the RTA. Here, the RTA decision is delayed because the underestimate of adverse winds leads to an optimistic time-to-get-home. The actual reserve endurance when arriving at base is only 110 minutes of fuel.

The final case shows an overestimated forecast of wind speed (60 knots) provided to the RTA. Here, the RTA decision to return-to-base is made prematurely because the RTA time-to-get-home estimate is expecting a strong adverse wind during the return trip. The actual reserve endurance when arriving at base is 25 minutes more than expected. Although returning to base early may be beneficial for the safety of the vehicle itself, prematurely leaving the mission will likely have other undesirable consequences.

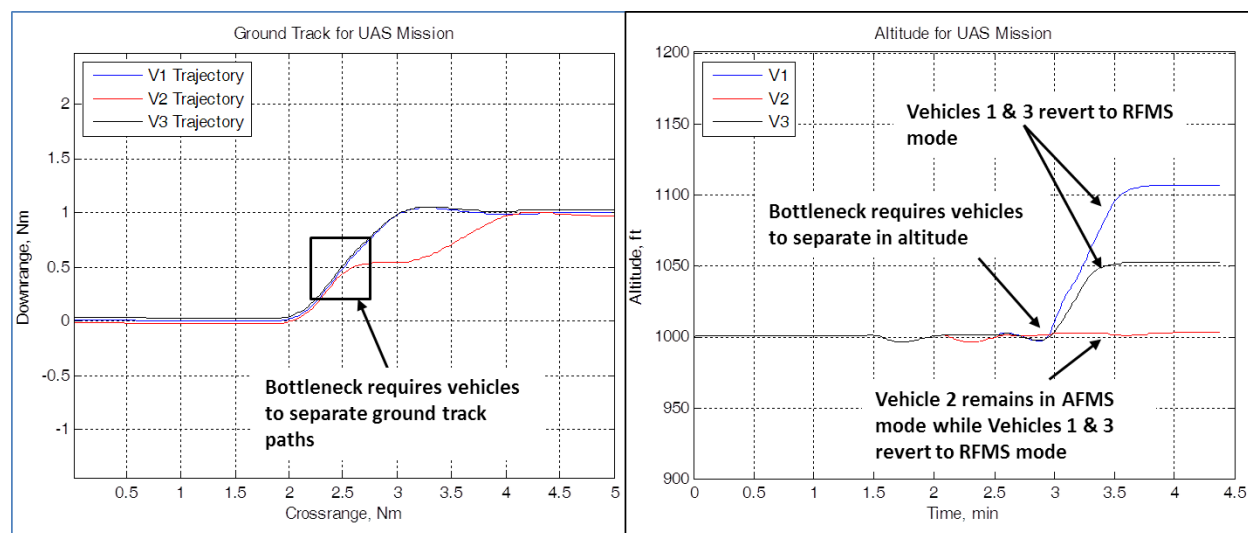
Built in margins in the FMS RTA decision process will be a function of the uncertainty in wind conditions, the importance of completing the mission, and the importance of safely retrieving the UAS equipment. The more value placed on completing the mission will require a less conservative reserve endurance threshold, whereas more value placed on safely recovering the equipment will require a more conservative reserve endurance threshold. The actual threshold value uploaded onto the onboard FMS RTA may be done during day-of-mission pre-planning when more accurate estimates of wind-aloft conditions can be utilized.



**Figure 58. Effects of Forecasted Winds-Aloft on FMS RTA Reserve Endurance Check**

### 7.6.3 FMS RTA Check on RSV Separation through Bottleneck

We also performed an initial investigation using our multi-vehicle simulation to study the RTA check on required RSV separations through a bottleneck experienced during the course of the mission. Figure 59 shows the results of this experiment. Here, we programmed the reversion procedure to change both ground track and altitude if required to maintain the proper distances of the RSVs for all vehicles. For this particular scenario, the interesting result was that two of the vehicles (vehicles 1 and 3) reverted to their RFMS's and then changed course and, in addition, changed altitude to ensure their RSVs were deconflicted. However, once this course of action was activated, the FMS RTA of vehicle 2 had no conflict and did not revert to its RFMS. It continued on with the mission at the nominal altitude and the nominal ground track course.



**Figure 59. Experimental Results of Required Fleet Separation due to Encountered Bottleneck**

Although the we did not pursue further experiments with this scenario, this highlights the unexpected results that can occur with multi-vehicle fleets. If the timing is not exactly coordinated between the vehicles in the fleet, then their FMS RTAs may not all make equivalent decisions regarding whether to revert or to remain in advanced mode. Once the bottleneck threat has passed, and presumably they attempt to rejoin, then some of the vehicles will be in advanced FMS mode, while others will be in RFMS mode. This is an example of yet another layer to the interactions that can be experienced by fleets of UAS platforms, each with their own nested, hierarchical set of RTA protected feedback loops. They cannot all be expected to experience the same situations over the course of the mission. As a result, in some cases there will be reversions made by the RTA monitors to the backup systems at particular feedback levels, whereas in other cases, there may be no need to revert. These potential disparate mode portfolios increase the complexity of allowing multiple nested RTA protected systems on interacting, distributed platforms in a cooperative command/control architecture. Further study of the implications of allowing this myriad of mode options during runtime is recommended.

## 8 Certified Collision Avoidance Integrated with RTA Protection

### 8.1 The Need for Certified Collision Avoidance

As discussed in the previous chapter, one of the main functions of the FMS and the FMS RTA is to ensure that the vehicles in the fleet maintain proper and safe separation distances at all times. Under nominal conditions, this should be the case, whether in advanced or reversionary modes at the FMS level. We consider this to be *path conflict resolution management*.

However, unforeseen events can always occur during the course of a mission requiring emergency, evasive actions to avoid collisions. This may occur due to an unknown ground obstacle that was not defined in the terrain information from the MPS level and is only detected by onboard sensors *at the last moment*. Or, a fleetmate's sensors or other hardware have stopped working correctly causing it to encroach into the ownship's RSV. Or, an intruder aircraft may enter into the mission path of the fleet. The intruder may be unaware of the fleet or may have hostile intent. Such a scenario has the potential to cause multiple collision avoidance tasks, where each vehicle in the fleet needs to avoid the intruder, but also avoid each other and any other obstacles in proximity. For these reasons, there is a requirement to have onboard sense and avoid or collision avoidance (CA) functionality. *We consider that collision avoidance is different than path conflict resolution management because collision avoidance requires the vehicles to do something different from their intended mission actions. Path conflict resolution management is a normal part of planning out the mission path for the fleet.*

Although there may be separate CA functions for different threats, such as ground or airborne, that may utilize different sensor hardware, we will group all CA functions together here and consider one universal CA function. The avoidance actions taken for different threats will typically utilize the same geometric maneuver primitives, although the maneuvers may be more or less aggressive depending on the type of threat being avoided.

At the beginning of this program, we considered that there may be advanced CA functionality that resides within the AFMS and AGLAW blocks. Here, the AFMS would operate threat determination functions, taking in available sensor information and resolving whether observed intruders or obstacles pose a threat and require avoidance actions. If this is the case, then the AGLAW would be responsible for generating an avoidance path. However, the idea of having advanced CA systems would require RTA monitoring and we eventually realized there is a fundamental flaw to this approach.

Collision avoidance is the single most critical safety issue for unmanned systems, especially when operating with manned aircraft or in populated theaters in which collisions could cause injury or loss of life. For this reason, we expect that certifying governing authorities will dictate that CA protocols require *maximum margins* when performing avoidance maneuvers. That is, avoidance actions should be taken as soon as a threat is determined and the vehicle should not wait until the *last moment* to avoid the threat.

However, if we consider advanced systems performing CA functions, then the RTA monitoring will need to allow these advanced CA systems to operate up to the *minimum margin* before reverting to trusted CA functions so as to not limit the operational envelope of the advanced CA elements. Yet, we expect that such a protocol would never be able to be certified for operation.

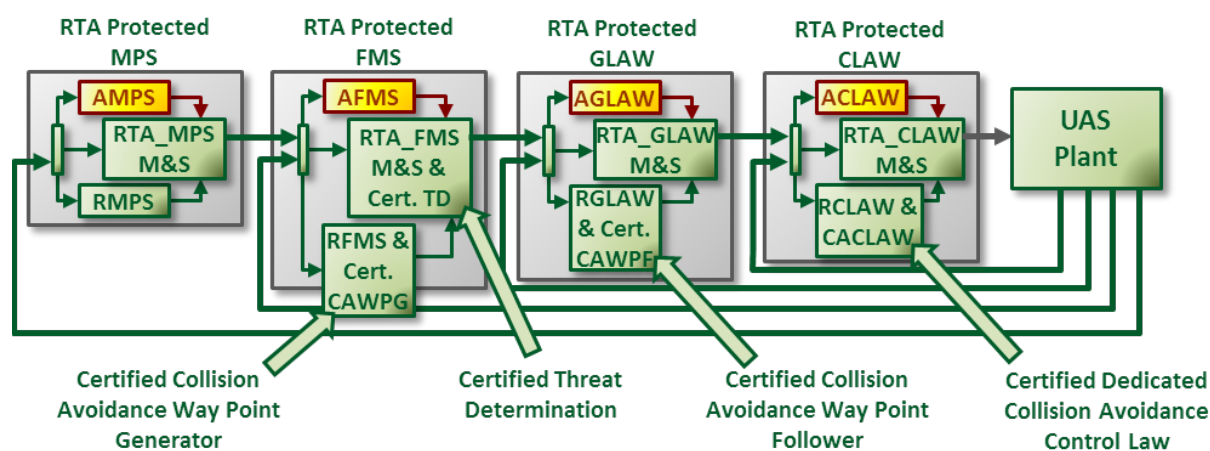


Instead, the RTA monitor would have to switch to the trusted CA system when the maximum margin is encountered, or when the threat is first determined. However, this negates any reason for considering the use of advanced CA systems, since trusted CA systems already operate in this manner. In summary, because of the criticality of collision avoidance, certification authorities would never allow for any type of advanced or minimum margin behavior from a CA function. Therefore, we only need to consider that trusted, certified CA systems will be housed on future UAS platforms.

## 8.2 Recommended Frameworks

### 8.2.1 Certified Collision Avoidance as an Integral Part of RTA

The Air Force is currently considering collision avoidance functionality to be a part of a larger assurance monitoring element for future advanced systems. This larger monitoring function would be analyzing the state of the system for software and/or hardware faults, collision threats, air traffic management redirection, etc. To align a framework under this consideration, we first recommend that the certified collision avoidance system (CCAS) elements be an integral part of the RTA elements. Figure 61 presents this recommended framework. This figure shows that the threat determination function is one of the monitoring functions within the FMS RTA monitor and switch block. If a threat is determined, then the certified collision avoidance waypoint generator is activated within the RFMS block. A certified collision avoidance way point follower is housed within the RGLAW block and a dedicated certified collision avoidance control law is housed within the RCLAW block. These may be considered the primary RTA monitoring functions and reversionary actions, given that collision avoidance is considered to be the primary safety concern for UAS platforms. Note, however, that a detected threat may not necessarily be the result of any error in an advanced system, and may simply be the fault of the intruder aircraft. Once the threat has been successfully avoided, the RTA systems should have the logic to be able to determine if they should switch back to the advanced systems, or continue with reversionary modes.

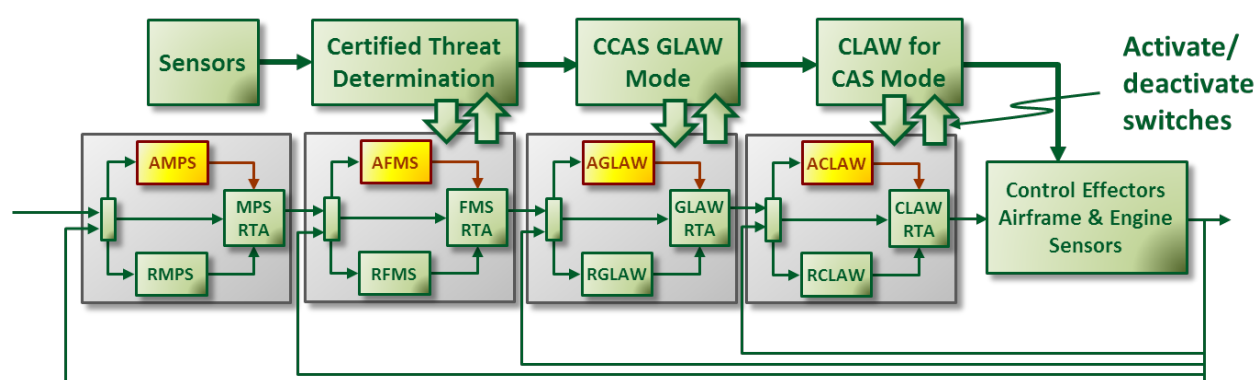


**Figure 60. Certified Collision Avoidance System as an Integral Part of RTA**

### 8.2.2 Certified Collision Avoidance Separate from RTA

For redundancy and further modularity, it may be desirable to house CCAS functions on separate onboard processors, for example, or at least as a separate set of code that executes outside the

RTA protected feedback loops. Figure 61 indicates the collision avoidance functions being housed on separate processors or residing outside of the RTA protected feedback loops. The certified threat determination function should operate continually, always checking for collision threats as information becomes available from situational awareness sensors. If an intruder or obstacle is determined to be a threat, normal operations are ceased and the CCAS GLAW and dedicated certified CLAW are activated to perform the appropriate avoidance maneuver. Once the threat has passed, then operations can be switched back to the RTA protected nested feedback loops. Again, note that if collision avoidance actions are necessary, we do not consider this to necessarily be an indication of a flaw in any of the advanced elements. Therefore, once the threat has passed, operation is returned to the advanced elements as long as they were active at the time the threat was first determined. Operation is returned to the reversionary systems if it is determined that collision avoidance activation was the result of an error in one of the advanced systems.



**Figure 61. Certified Collision Avoidance System Separate from RTA Framework**

However collision avoidance is to be integrated, whether as an integral part of the RTA elements, or as a separate set of functions, collision avoidance is such a safety critical function it will need to be certified and completely trusted to perform correctly at all times during the mission.

The rest of this chapter presents an overview of collision avoidance and details on a collision avoidance approach that should be able to be certified at design time.

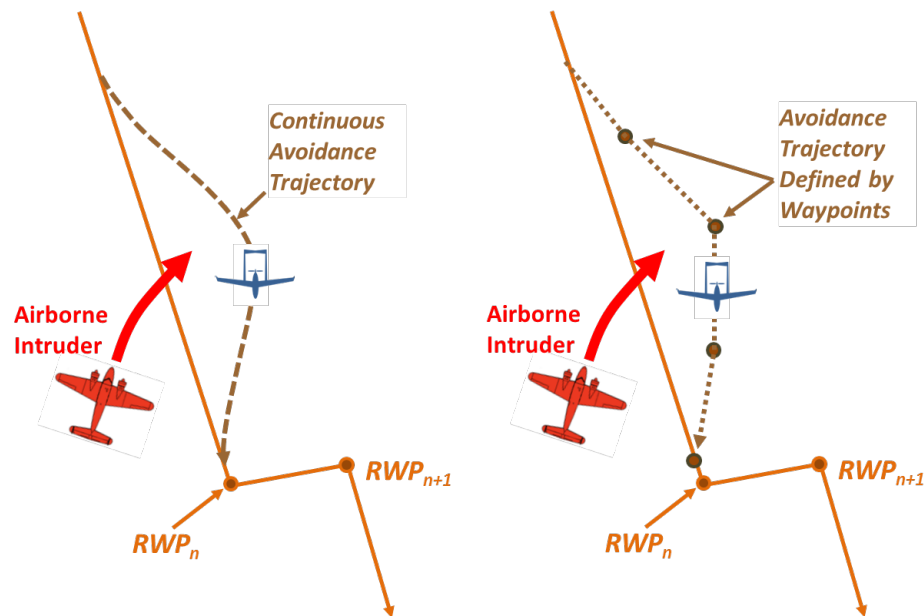
### 8.3 Overview of Collision Avoidance

There are three main functions in collision avoidance procedures:

1. Sensing function: potential hazards are detected through onboard sensors, such as EO/IR cameras, radar, transponders, ADS-B, etc. Sensing for situational awareness is assumed active at all times. Although the sensor equipment may be used more for surveillance during flight over certain locations, there will always be some amount of time over a given set of update frames in which these sensors are used for detecting neighboring potential hazardous objects.
2. Threat determination: once nearby potential hazards are detected, algorithms are needed to determine if they pose a potential threat. This is typically done by predicting both the

ownship's trajectory and the threat's trajectory (if a dynamic threat) over a certain horizon time. The nearby object may be a fleetmate that is flying parallel to the ownship, therefore their predicted trajectories will not intersect, and the fleetmate is not classified as a threat. However, if the two trajectories intersect a "buffer zone" around the ownship (usually a cylinder of a defined radius/height) then the object is classified as a threat.

3. Collision avoidance: once a threat (or multiple threats) is identified, collision avoidance algorithms determine avoidance trajectories to resolve the conflict. These may include lateral maneuvers for changes in heading, pitch up or pitch down maneuvers for changes in altitude, or changes in airspeed. The avoidance trajectories are delivered to the guidance system to follow. They can be either in the form of continuous paths, or *quantized* as a set of waypoints. This is illustrated in Figure 62.



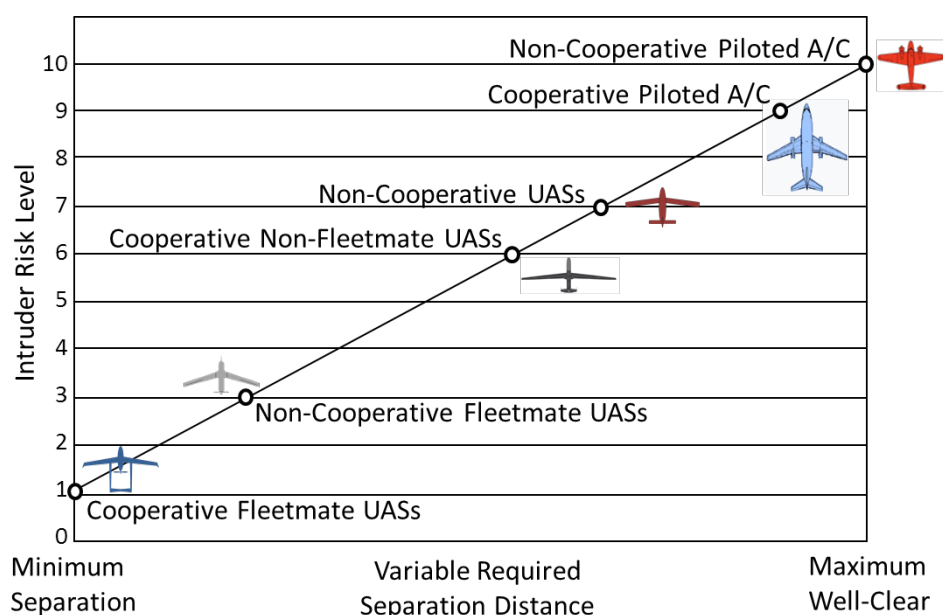
**Figure 62. Avoidance Trajectory Types: Continuous or Waypoints**

### 8.3.1 Classes of Collision Avoidance Functions

We consider the following classes of collision avoidance functions:

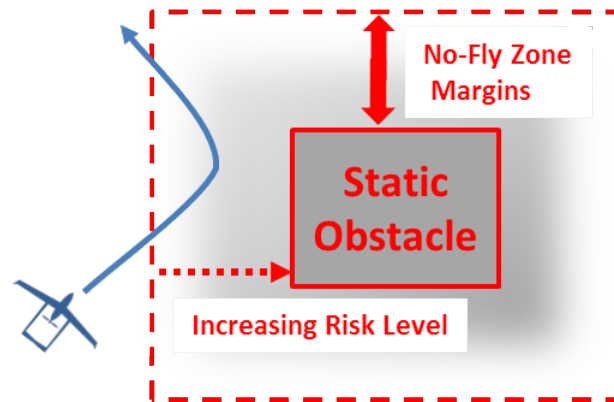
1. Cooperative dynamic avoidance: here, aircraft broadcast their state information to each other for purposes of deconflicting their paths. Since each vehicle will take the appropriate avoidance actions, their separation distances can usually be allowed to be relatively small, depending on the types of aircraft (piloted or unpiloted) and the rules of the air in that particular theater. For fleets of UAS vehicles, we assume each would utilize cooperative collision avoidance with all neighboring vehicles as long as their communication hardware is working properly. We assume here that vehicles in the fleet are allowed to fly at a defined *minimum separation distance* from each other. However, for cooperative piloted aircraft, for safety reasons the separation distance will be required to be much larger, and this is often referred to as the *well clear* distance.

2. Non-cooperative dynamic avoidance: in contested theaters there may be instances in which intruder aircraft are unaware of the ownship's path and are not actively communicating or broadcasting their state. Or, a fleetmate with a nonworking transmitter, unable to broadcast its state would be classified as being non-cooperative simply because two-way communication is absent. In these cases, the ownship must execute more extensive avoidance maneuvers resulting in larger separation distances. This may again be the well clear distance, or even a larger distance since the intruder is non-cooperative and its future state can only be predicted. A risk value may be assigned to intruders if target identification can be performed and the value of the well clear distance may be a function of the risk level of the intruder. For safety reasons, the well clear distance may be largest for piloted intruder aircraft, and smaller for intruder UASs, for example. Figure 63 illustrates how the required separation distance may be defined as a function of the risk level for various types of intruders. The separation distance may also be a function of the intruders closing rate, for example.



**Figure 63. Variable Avoidance Distance as a Function of Risk**

3. Static avoidance: in contested areas there will be no-fly zones due to ground based threats, such as enemy anti-aircraft batteries, elevated terrain, urban obstacles, etc. Enemy ground vehicles may be moving slow enough relative to the aircraft's airspeed to be considered static threats. Encounters with static obstacles will also require appropriate avoidance maneuvers. No-fly zones may define distance margins around static obstacles, as illustrated in Figure 64. Under nominal conditions, all UASs should completely avoid the outermost boarder (red dashed line). Yet, due to unfolding events during the mission, such as bottlenecks with other fleetmates, an ownship may be required to enter the no-fly zone. However, the farther inside the no-fly zone, the greater the risk.



**Figure 64. No-Fly Zone Margins as a Function of Risk**

Table 5 presents a list of important distances for collision avoidance, with example numerical values, common for UAS systems. The required separation distances are assumed to be defined with enough margin for successful and robust collision avoidance in the face of modeling uncertainties and environmental disturbances, such as winds/turbulence. The capabilities of the sensor equipment in detecting intruder threats can be a main contributing factor on how required separation distances are defined.

**Table 5. Defined Distances for Collision Avoidance Function**

| Distance                              | Description                           | Value            |
|---------------------------------------|---------------------------------------|------------------|
| R1                                    | Fleetmate minimum separation distance | 800 ft           |
| R2                                    | Intra-fleet communication range       | 2,500 ft         |
| R3 <sub>min</sub> – R3 <sub>max</sub> | Intruder well clear distance          | 3,000 - 8,000 ft |
| R4                                    | Camera observation range              | 10,000 ft        |
| R5                                    | Radar observation range               | 40,000 ft        |
| $R1 < R2 < R3 < R4 < R5$              |                                       |                  |

### 8.3.2 One General Collision Avoidance Algorithm

Again, it is assumed here that the CA function is one general algorithm that generates the avoidance procedure. The encounter scenario will define the numerical values used and the specific protocol, but these will be inputs into the general CA system. Whether the ownship is avoiding a cooperative fleetmate, a non-cooperative airborne intruder or a ground based object with a no-fly buffer zone, it will either perform a change in its heading, altitude, or airspeed, or a combination of the three. This is accomplished through either a coordinated banked turn in the proper direction, climbing or descending in altitude through a pitch up or pitch down maneuver to change the vehicle's flight path angle, or throttling up or down to change the airspeed. The separation distance to be maintained is simply a function of the hazard and the rules of the air in that particular theater of operations. Although collision avoidance functions today are separate systems (ACAS, TCAS, AGCAS, for example), we assume here that far future UAS platforms will have all collision avoidance functions in one single, general system.

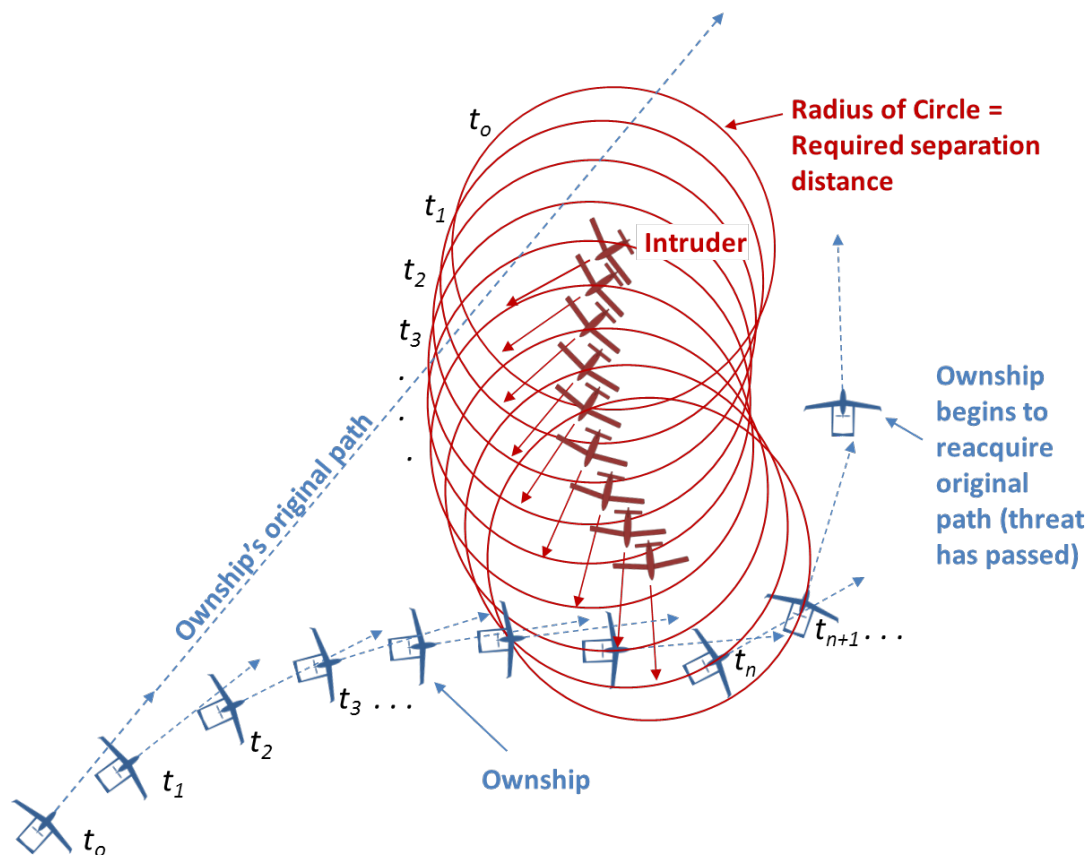
In [Cooper 2014], an architecture is presented whereby the avoidance actions taken by the vehicle are largely dictated by *user defined strategy profiles*. Due to the performance characteristics of typical aircraft, it is usually desired that the avoidance action should first be a lateral maneuver (i.e., a coordinated banked turn) in the proper direction. Altitude changes (either climbing or diving) or airspeed changes should only be performed if necessary to avoid multiple intruders. However, in some theaters it may be preferable to change altitude first, rather than perform a lateral maneuver. Or, first changing velocity may be dictated in some applications. We developed this general architecture 1) because no governing body has yet dictated requirements for UASs to avoid other vehicles, and 2) so that our collision avoidance system is made for general multi-theater use.

#### 8.4 Certified Reactive Collision Avoidance

Threat determination is an active area of research for development of collision avoidance systems. Some approaches have considered complex prediction schemes based on predicted achievable footprints of the intruder. We took a much more straightforward approach in our MUSAA program [Cooper 2014] and believe this approach can be successfully certified at design time with current V&V methods.

The approach developed in the MUSAA program is known as *reactive collision avoidance*. At each update the algorithm takes in the intruder's current position and velocity from the sensor block and predicts a straight line path forward assuming airspeed is constant at its current sensed value. For the case of lateral avoidance maneuvers, if the intruder turns toward the ownship, the ownship reacts by changing its heading more away from the intruder's path. If the intruder turns away from the ownship, the ownship reacts by changing its heading back toward its original planned path. In this manner, deviations from the ownship's nominal course are minimized. *This approach works well because avoidance maneuvering begins as soon as the intruder is determined to be a threat and the ownship has a reserve of its maneuvering footprint.* Under nominal conditions the ownship should be able to maintain the required separation distance, which it will target as the closet point of approach. As such, it will *ride along* the circle centered at the intruder's current position with radius equal to the required separation distance. Again, we denote this as the *required separation volume* (RSV). Once the intruder has passed, the ownship directs its heading back to its original path. Figure 65 illustrates a case with a series of *time stamps* (labeled  $t_o, t_1, \dots, t_n, t_{n+1}, \dots$ ) in which the intruder continually turns toward the ownship. The ownship reacts by continually turning away from the intruder until it rides along the boundary of the RSV. Once the intruder has passed, the ownship begins to reacquire its original path.

The collision avoidance system developed in our MUSAA program also had a contingency mode if there were a case in which the intruder was not detected until it was closer than the required separation distance. In that case, the ownship is commanded to its maximum bank angle directing its heading away from the intruder until the required separation distance is achieved. Then, the nominal reactive collision avoidance procedures are activated.



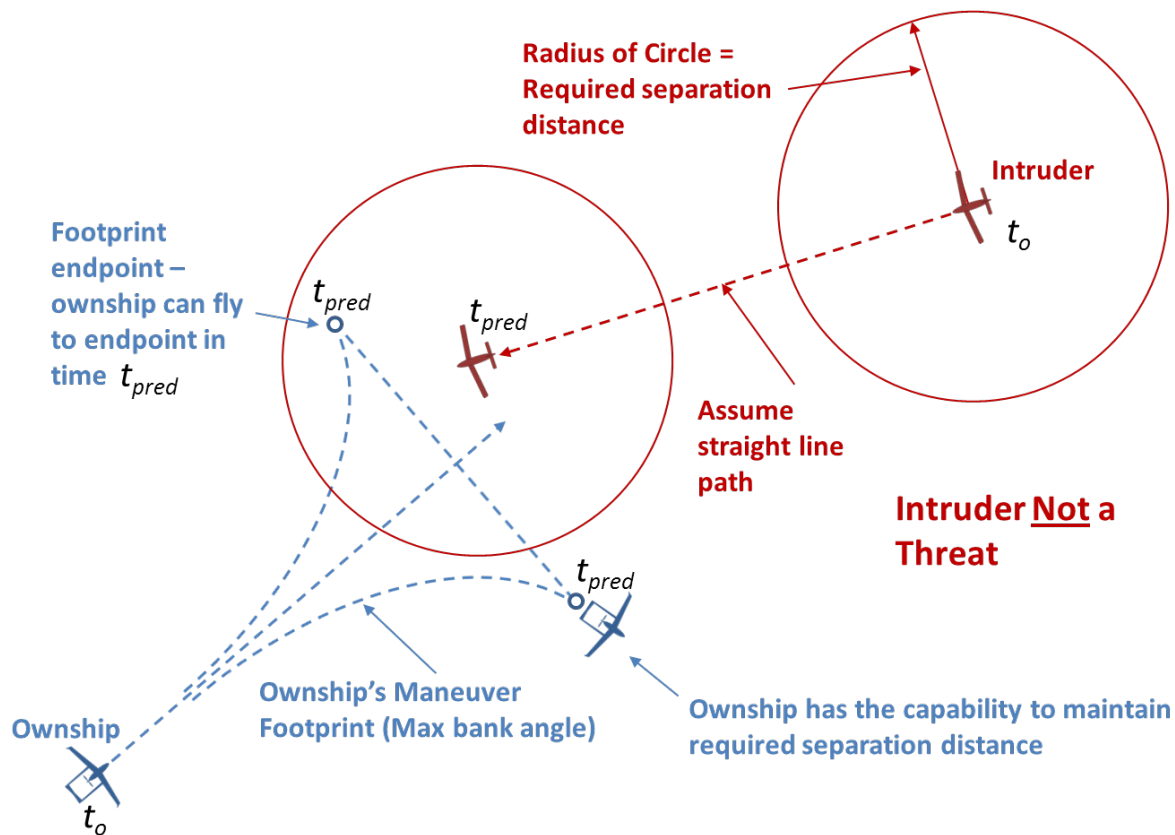
**Figure 65. Nominal Reactive Collision Avoidance**

If threat determination is a check performed by the FMS RTA, we would not have the luxury to begin avoidance maneuvering at the first determination the intruder poses a threat (when typically there is a reserve of maneuvering capability to avoid the intruder). Otherwise, the FMS RTA system would be limiting the performance of the AFMS in planning out an advanced path, which may very well correctly avoid the intruder. In this case, the threat determination function must determine the *last possible time* in which to activate a certified collision avoidance function, possibly residing within the RFMS, or in an outside loop, as shown in Figure 61. There is not, however, a unique way to make that determination and it depends on the intruder prediction model used. For demonstration purposes, we discuss a simple approach here, which is a variant of the reactive collision avoidance approach. This approach illustrates a *minimum margin CA* approach. The steps taken are as follows:

1. Determine the ownship's maneuvering footprint based on maximum bank angle and airspeed capabilities.
2. Estimate intruder's current position, airspeed and heading from sensor measurements.
3. Predict the intruder's future trajectory assuming a straight line path and constant current airspeed.
4. Check to see if some part of the ownship's maneuvering footprint lies outside of the required separation circle, centered at the intruder's predicted future position. If so, the ownship has a reserve of maneuvering capability to avoid the intruder. The FMS\_RT A should not activate



certified collision avoidance. This case is shown in Figure 66. The intruder is projected in a straight line from its position at the current time  $t_o$  to a predicted future position at time  $t_{pred}$ . The ownship's maneuver footprint is also shown from the current time to the same predicted future time. This shows all the possible positions the ownship is capable of reaching in that amount of time, given its current maneuvering and airspeed capabilities. The left and right footprint endpoints are the farthest points in space the ownship can fly in the time  $t_{pred}$ . It can be seen that as long as the intruder maintains a straight line path, then the ownship has the capability of maintaining a distance equal to or greater than the required separation distance since the right end point of the maneuver envelope is outside the required separation circle.

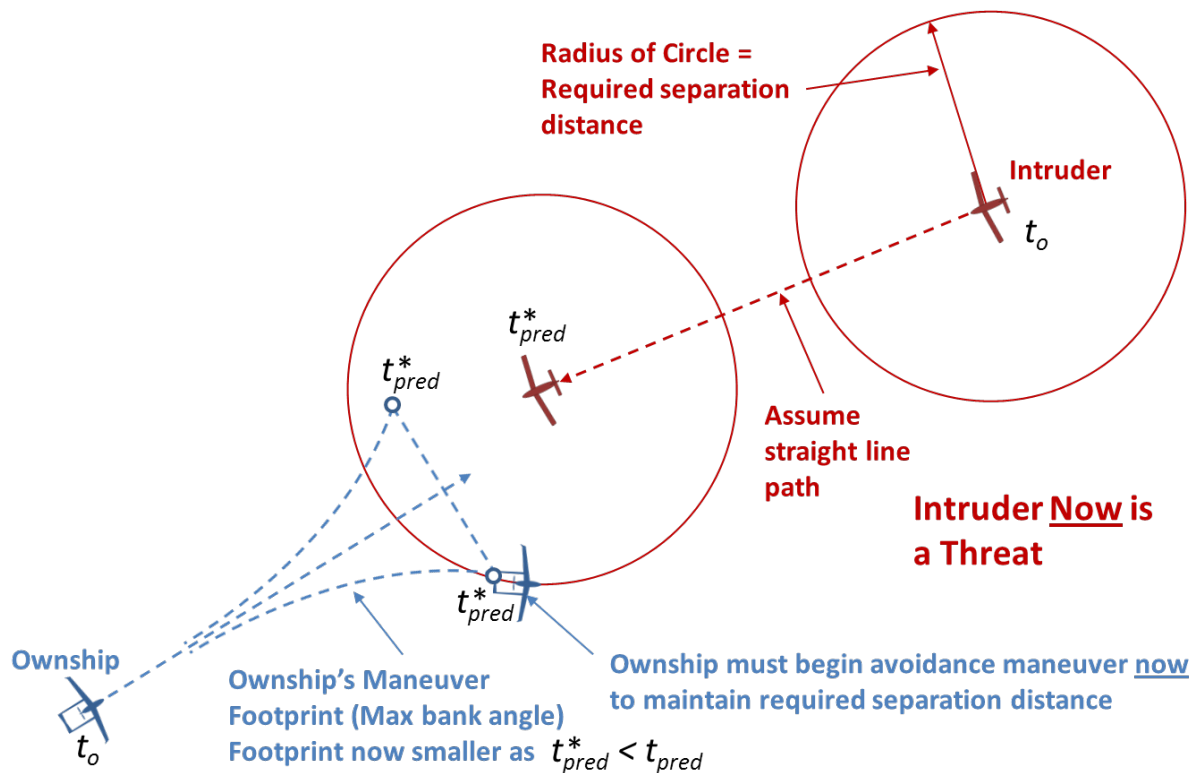


**Figure 66. Predicting Ownship's Capability to Maintain Required Separation – No Threat**

5. As the intruder flies toward the ownship, its closing distance continues to get smaller. The above prediction steps are repeated at each update of the FMS until either a) the closing distance begins to grow larger (i.e., the threat has passed) and the ownship always has the maneuvering capability to maintain the required separation (thus the AFMS is not shut down and reversion is not activated), or b) at some future time, neither endpoint of the maneuver envelope lies outside the required separation circle. Either both endpoints lie on the circle (the case of a head on approach by the intruder) or one endpoint lies on the circle and the other endpoint is inside the circle. This implies the ownship no longer has a reserve of maneuvering capability to maintain the required separation and the FMS RTA must shut down the AFMS and activate the certified collision avoidance function. This case is shown



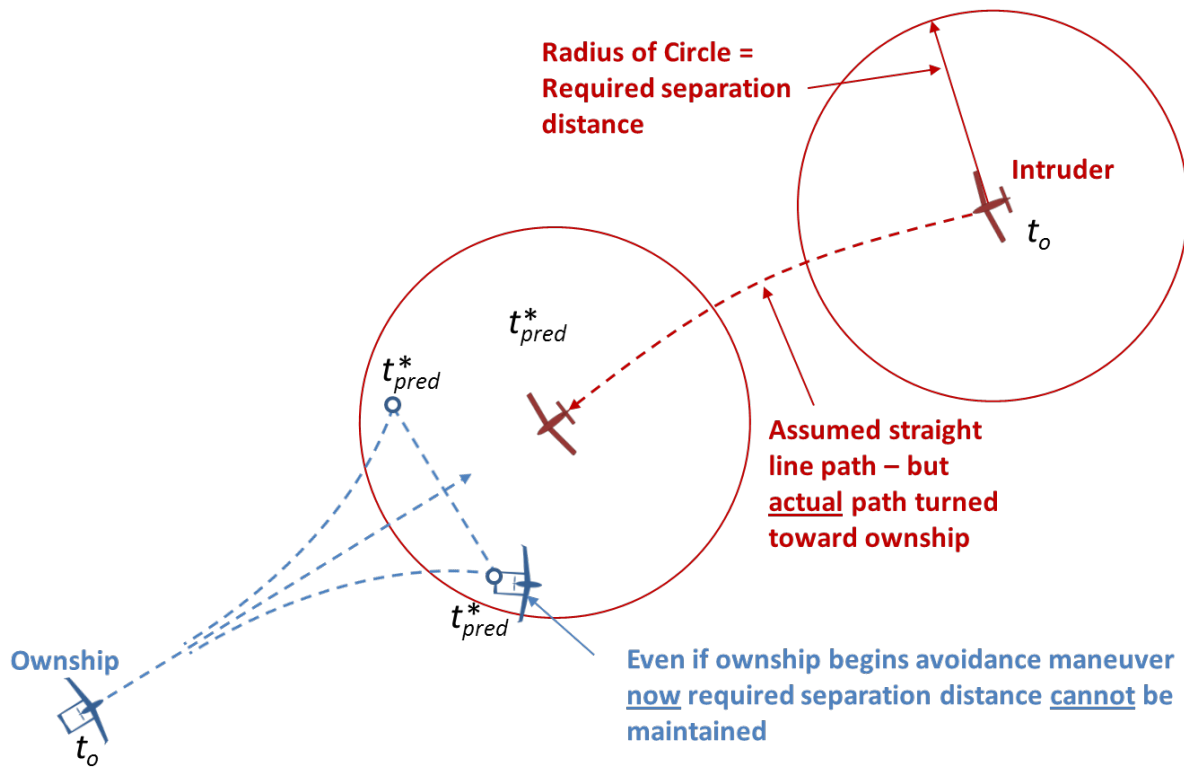
in Figure 67. This shows a case where the engagement is more head on than shown in Figure 66, and the vehicles are closer in range hence the closing time is smaller (hence the prediction time is smaller:  $t_{pred}^* < t_{pred}$ ) and the ownship's maneuver footprint is smaller (it cannot cover as large an area in a smaller amount of time). Therefore, once the certified CA is activated, it must first command the maneuver depicted by the right edge of the ownship's maneuver envelope so that at the future time  $t_{pred}^*$  it will be flying at the required separation distance from the intruder (flying on the required separation circle with heading tangent to the circle).



**Figure 67. Predicting Ownship's Capability to Maintain Separation – Threat Identified**

6. At this point, the certified CA function can execute the reactive collision avoidance procedure, depicted in Figure 65 and ride the edge of the required separation distance until the intruder has passed.
7. One key problem with this approach is that it assumes the intruder will fly a straight line path to the point shown in Figure 67. If in reality, the intruder actually turns toward the ownship during the prediction time horizon, then the ownship will not have the maneuvering capability to always maintain the required separation distance, and will be inside the circle at the future time  $t_{pred}^*$ . This case is illustrated in Figure 68. This can potentially be mitigated because the intruder's initial heading used in predicting its future path is updated at each update to the FMS\_RTA. Therefore, by continually updating the straight line predictions, the curvature in the intruder's path will be captured, assuming the filtering of the sensor measurements can give accurate estimates of the intruder's state. Further, flying the edge of

the ownship's maneuver envelope is equivalent to the maximum bank angle maneuver commanded in the contingency mode developed in our MUSAA program.



**Figure 68. Incorrect Prediction of Intruder's Path – Required Separation Not Maintained**

Nevertheless, this approach highlights its non-conservative nature, allowing the advanced system to operate until the ownship must perform an aggressive maneuver to avoid the threat, which could result in a breach of the required separation distance if the intruder performs some unexpected maneuvering. It is for this reason that most likely advanced CA functionality will simply not be allowed in future unmanned systems. The CA process should be conservative, and avoidance maneuvering should begin as soon as the intruder is determined to be a threat. This then, obviates the need for advanced CA systems, which obviates the need for RTA monitoring of such systems.

## 9 RTA Protection Applied to Mission Planning Systems

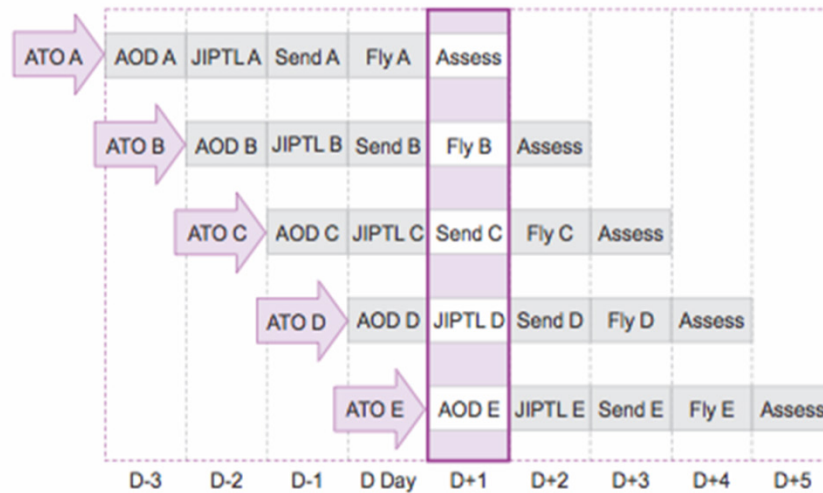
### 9.1 Current Mission Planning Protocols

Since mission planning of UAS missions can be a complex, detailed task, we sought information on how such activities are carried out currently. However, due to the sensitive nature of military planning of UAS operations, there is little open source information. However, in [Joint Chiefs of Staff, 2014], the protocols for joint air operations are reviewed and cover both manned and unmanned missions. This publication provides joint doctrine for the command and control of joint air operations across the range of military operations. The following are important excerpts from this document:

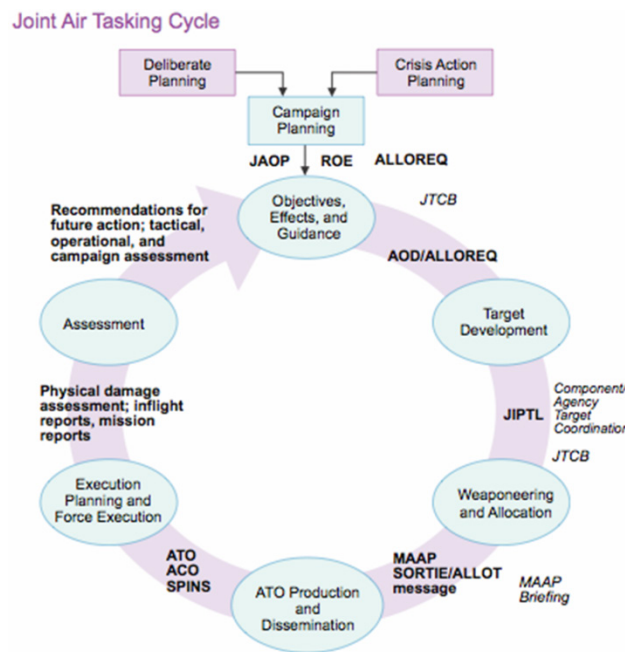
- Joint air operations are normally conducted using centralized control and decentralized execution to achieve effective control and foster initiative, responsiveness, and flexibility.
- Centralized control facilitates the integration of forces for the joint air effort and maintains the ability to focus the impact of joint air forces as needed throughout the operational area.
- Decentralized execution is the delegation of execution authority to subordinate commanders.
- The intent of mission command is for subordinates to clearly understand the commander's intent and to foster flexibility and initiative at the tactical level to best accomplish the mission.
- Over-controlling joint air operations can reduce tactical flexibility, taking away initiative from operators.
- Insufficient guidance may result in failure to capitalize on joint force integration or may degrade operational-level flexibility, such as shifting Joint Force Commander (JFC) priorities, thus reducing effectiveness.
- The Joint Force Air Component Commander's (JFACC's) planners must anticipate the need to make changes to plans (e.g., sequels or branches) in a dynamic and time-constrained environment.
- The mix of friendly, adversary, and neutral aircraft and mission constraints may require the JFC to strictly control flights in the operational area.
- The Air Tasking Order (ATO) articulates the tasking for joint air operations for a specific execution timeframe, normally 24 hours.
- The Joint Air Operations Center (JAOC) normally establishes a 72- to 96-hour ATO planning cycle.
- The battle rhythm is essential to ensure information is available when and where required to provide products necessary for the synchronization of joint air operations with the JFC's CONOPS and supporting other components' operations.
- Nonetheless, airpower must be responsive to a dynamic operational environment and the joint air tasking cycle must be flexible and capable of modification during ATO execution.

The following two figures are reproduced from this document and show the battle rhythm and air tasking cycle. Planning and execution may span multiple days and extensive planning, analysis, and wargaming are typically performed before tasking begins.

### Joint Air Operations Center Air Tasking Order Battle Rhythm



**Figure 69. The Battle Rhythm – Reproduced from (Joint Chiefs of Staff, 2014)**



**Figure 70. The Joint Air Tasking Cycle - Reproduced from (Joint Chiefs of Staff, 2014)**

Additional excerpts from this document are:

- During execution, the JAOC is the focal point for changes to the ATO and is the centralized control node for tasking of joint air capabilities and forces. It is also charged with coordinating and deconflicting those changes with the appropriate control agencies and components.
- Retasking an airborne ISR asset during mission execution must be carefully considered.

- Dynamic retasking of ISR assets should be done by the appropriate commander after evaluating the full impacts of diverting the capability from the current mission and the impact to operational success or consequences without the asset.
- UASs should be treated similarly to manned systems with regard to the established doctrinal warfighting principles.
- There are three categories of UAS tasking:
  - Preplanned. UASs tasked by the JFACC.
  - Immediate. Requests are submitted outside the ATO cycle and expedited through internet relay chat, email, telephone, or radio, as required. Immediate requests are sent directly to the JAOC.
  - Dynamic. Redirects UASs from an existing mission to a new target based on published priorities and criteria. Historically, reasons for retasking include troops in contact, high priority target opportunities.
- Retasking a UAS must be carefully considered.
- Dynamic retasking of a UAS should be done by the governing C2 agency and only after evaluating the full impact of diverting the capability from the current mission and the impact to operational success or consequences without the asset.

Our main conclusions from this information are that 1) mission planning is an extensive, detailed activity, spanning several days and several agencies, involving multiple personnel, 2) the mission plan is the Air Tasking Order (ATO), which has an initial plan, generated before mission start, but 3) the ATO can be changed during the mission to allow for flexibility, and 4) however, major changes must be carefully considered and approved by ground base commanders (humans in the loop).

Autonomous mission planning of UAS swarms has gained attention in recent years, with a number of approaches investigated with application to urban reconnaissance, environmental data gathering, border patrol, etc. [Torens 2013], [Ke 2009], [Boskovic 2009], [Karaman 2008], [Sullivan 2004], [Protti 2007], [Rubio 2004], [Stenger 2012], [Lamont 2008], [Kingston 2009], [Marshall 2011]. Our focus here will be on Air Force applications of UASs in contested, active battlefield arenas, performing, for example, intelligence, surveillance, and reconnaissance (ISR) missions, delivering supplies to forward units, attacking enemy forces with air-to-ground weapons, or providing air escort for high value air or ground assets (VIP missions) [Humphrey 2012], [Humphrey 2013].

## 9.2 General Description of MPS Functionality

For our fleet of UAS missions we consider that the main function of the onboard MPS is to execute the ATO. Although we have yet to find any detailed descriptions of an example ATO (again, most likely due to the sensitive nature of such information), we have attempted to model a simplified version based on the background given in [Joint Chiefs of Staff, 2014]. For the purposes of demonstration, we will focus on an ISR mission, in which the fleet takes off from a *home airbase* and flies to a set of objective locations to perform surveillance and intelligence data gathering. However, the framework is developed generally enough to be applicable to other UAS missions with the plan to use and extend the models developed in this project for follow-on Air Force efforts.

### 9.2.1 Command/Communication Architecture

We consider a decentralized, but cooperative command/control/communications (C3) architecture for the fleet. We assume radio communication range for the vehicles is limited to minimize the probability of detection as well as hardware limitations on signal strength. As such, we cannot assume all vehicles will have continuous communications with all other vehicles in the fleet or with a ground command/control center(s). Further, we do not assume the fleet is flying in formation with a “fleet leader” that commands the mission, with all other vehicles as followers, performing station keeping with respect to the leader. Each vehicle is responsible for carrying out its assigned mission. For these reasons, a decentralized architecture will allow for a feasible, robust planning solution. We also consider that the MPS can accommodate a heterogeneous fleet with varying assets, varying performances and varying numbers of vehicles, which can enter or leave the fleet during the course of the mission. Note that in a related program [DeVore 2014] we investigated the option of having the fleet *vote* for a fleet leader that would then make all mission decisions. This is an alternate option that can be further explored in follow-on efforts.

Since this is a decentralize command and communication architecture, the vehicles in the fleet will *negotiate* on a solution to the mission plan at a certain update rate, based on any new information or changes to the environment, scenarios, or mission objectives. Each vehicle should have an identical instantiation of the MPS. If this is not the case, different versions of MPSs should be certified to have compatible interfaces and compatible functionalities.

To draw an analogy to current industry standards, the MPS would be equivalent to autonomously generating a *flight plan*. Flight plans are typically developed manually by a pilot for general aviation (GA) aircraft, using maps and certified manuals, or drawn from a “library” of flight plans by air traffic managers for commercial aircraft. The FMS acts like industry standard flight management systems which are the onboard navigation systems that automatically follow flight plans. However, since we are exploring advanced capabilities for autonomous *thinking* fleets of vehicles performing military missions, the MPS and FMS will have additional functionalities beyond their counterparts in GA and commercial transport aircraft.

This framework should be designed so that any vehicle can operate on its own for single ship missions, but is also extensible so that vehicles can be easily *plugged into* any mission involving multiple vehicles. Initialization loads (I-loads) into the onboard processors that house the MPS/FMS will completely define whether it is a single ship mission or a multi-vehicle mission (i.e., no changes to onboard hardware or software are required to accommodate single or multi-vehicle missions).

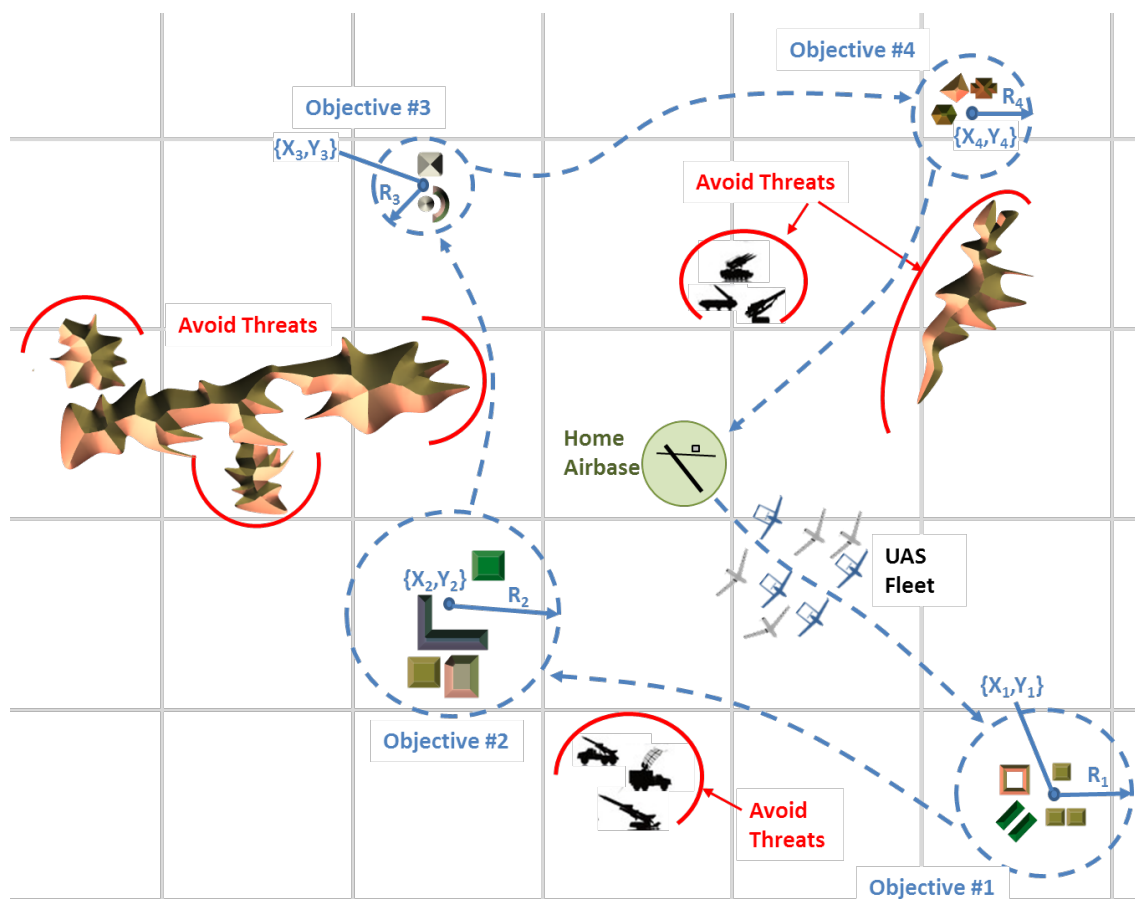
### 9.2.2 Initial Mission Planning – The Air Tasking Order

We assume that an initial mission plan for the entire fleet is developed at the ground base by mission planning personnel through the air tasking cycle process (see Figure 70). Let us assume this activity takes on the following protocol:

1. Pre-Flight Mission Planning: This activity develops a *pre-flight plan* or an initial ATO from overarching goals and directives from the command center. The UAS inventory is reviewed and vehicles are selected for the mission based on vehicle data, current conditions, and

performance capabilities (capable range, airspeed, etc.). For ISR missions, we assume *objective locations* are identified using the latest intelligence and terrain data. A *timing plan* is also developed, defining required arrival times to the *objective locations* and the amount of time allotted to surveil each location. Ground threats to be avoided are identified, such as mountainous terrain, enemy installations, or tall objects, such as buildings or towers. Assets are then allocated to specific tasks based on current risk/reward assessments that define risks in traveling to the *objective locations*, reward values for successfully completing the surveillance of each *objective location* and the values of each UAS.

An example of an initial ATO is shown in Figure 71, in which four UAS\_A vehicles and four UAS\_B vehicles fly from a home airbase to four *objective locations* to be surveilled. If they exist, the fleet may follow closely along roads connecting the *objective locations*. The figure shows mountain ranges and military installations to be avoided.

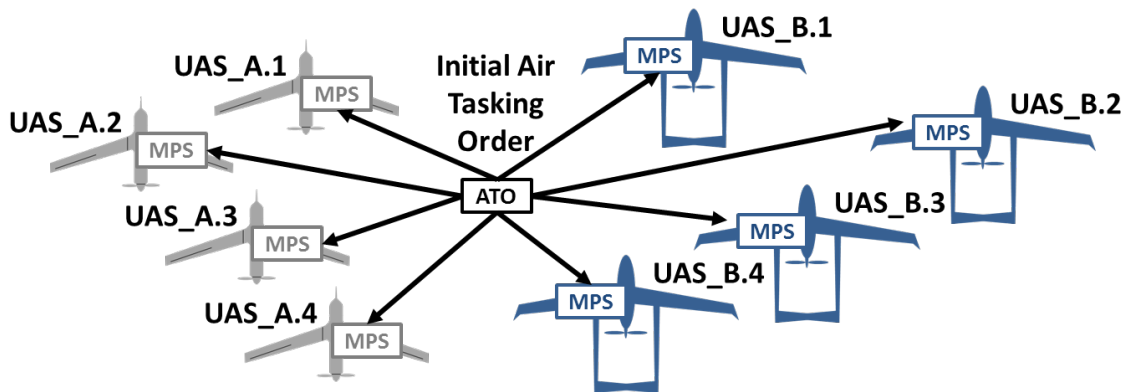


**Figure 71. Example Initial ATO for ISR Mission**

2. Ground Based Initial Mission Planner: We assume the mission planning personnel may be able to make use of ground based mission planning software that takes as inputs the specifics of the initial ATO and solves for additional information to be used onboard the vehicles:
  - a. Asset Allocation Plan (AAP): the AAP defines the order of each vehicle during the mission and any other required placement or assignment information. Although we did

not fully explore what information is included in the AAP in this project, we assume this information is needed onboard the vehicles during operation.

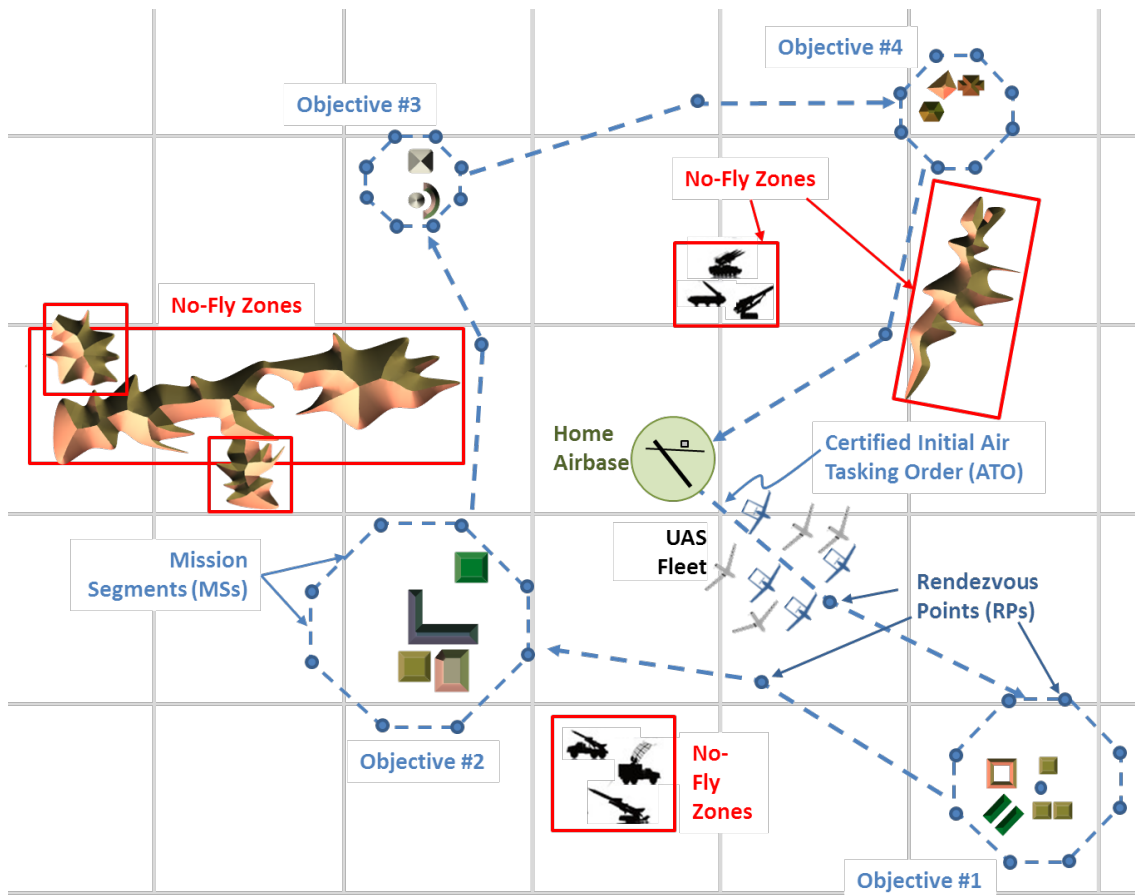
- b. Rendezvous Points (RPs): as previously discussed, RPs are locations for the fleet as a whole, each define by a  $\{\text{downrange, crossrange, altitude}\} = \{X,Y,H\}$ . It is assumed the fleet can retreat to any previous RP and loiter in a circular manner around the RP until such time it is deemed safe to continue the mission or call off the mission and return to the home airbase. The RPs also serve as locations to regroup and share information if some subset of vehicles becomes separated from the rest of the fleet.
  - c. Mission Segments (MS): MSs are straight line paths that connect each RP.
  - d. Timing Plan: this consists of required arrival times to each RP location for each vehicle in the fleet. Taken as a whole, these local timing plans for each RP should achieve the overall mission timing plan dictated in the ATO.
  - e. Terrain and other environmental information: we assume that pertinent terrain information is made available to the vehicles so that their FMS's can plan out local paths while avoiding threats, no-fly zones, etc.
3. Design Time Assurance of initial ATO: We also assume that DTA software may be available to V&V the initial ATO. Any iterations and changes to the plan are performed at this stage and the final plan is then reviewed by the appropriate planning personnel and certification authorities.
  4. Initial ATO uploaded to fleet: The certified initial ATO is then loaded to each vehicle's instantiation of the MPS during pre-mission preparations, as illustrated below.



**Figure 72. The Initial Air Tasking Order Shown Loaded to Each Vehicle's MPS**

Figure 73 presents the initial ATO for the ISR mission presented in Figure 71. The figure shows the RPs, MSs and no-fly zones defined to achieve the objectives defined in the *pre-flight ATO plan*. Again, for each MS, a prescribed airspeed for each vehicle is defined to achieve the *timing plan*.





**Figure 73. Example Certified Initial ATO for the ISR Mission**

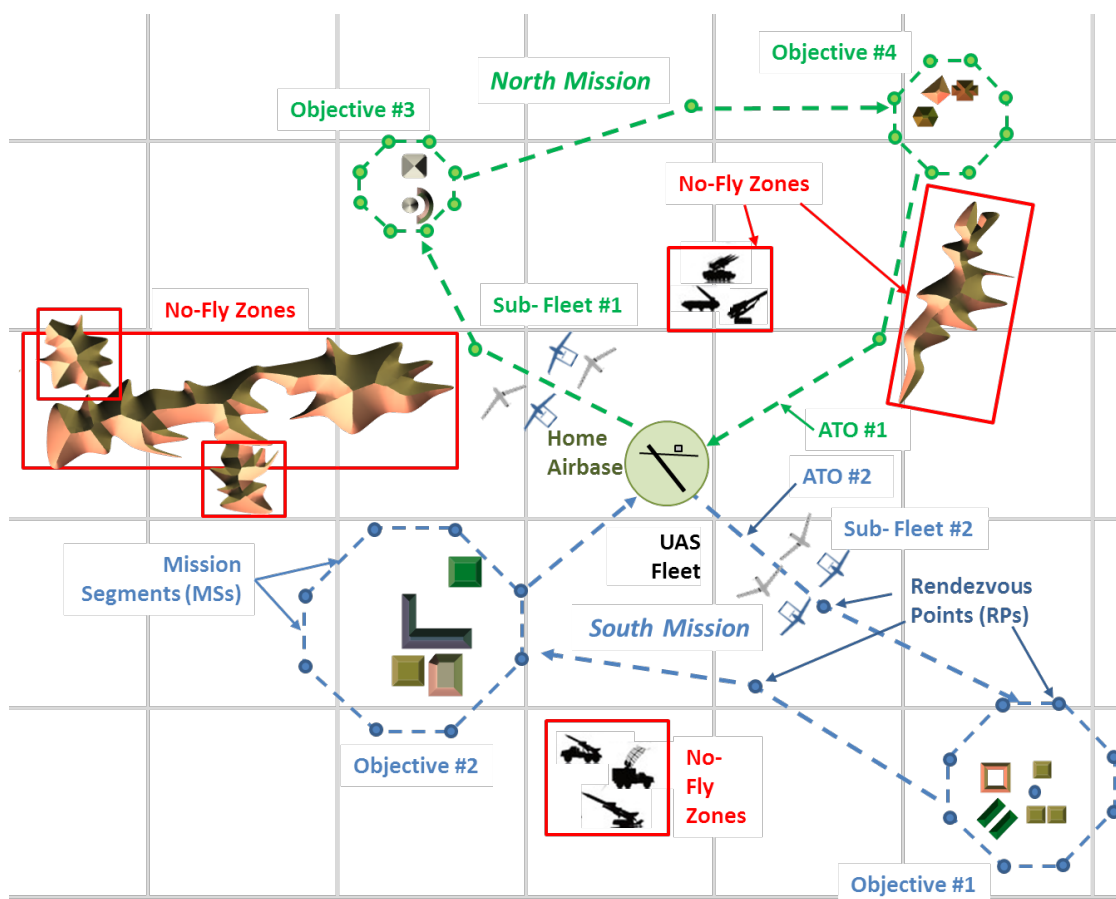
If needed, the initial ATO may be divided into two or more missions for allocated sub-fleets. Figure 74 presents an example of such a case in which the initial ATO is divided into North and South missions. However, we did not explore dividing missions in this project or scenarios where parts of the fleet break off to perform some other mission. Such case can have important implications to RTA checking and we leave this for future studies (see Chapter 13).

### 9.2.3 Difference between Advanced and Reversionary MPS Blocks

Again, because of our modular framework, the input/output structure of the AMPS and the RMPS should be the same. Since the main function of the MPS is to carry out the ATO, both these systems must be able to do so. Therefore, we define the RMPS as a certified software element that autonomously carries out the ATO at runtime, during the mission. The RMPS is considered to be conservative in that it only carries out the ATO plan and nothing more. Any variations to the mission will be approved by personnel at the ground command station. The RMPS will not have the capability to deviate from the tasks defined by the ATO unless such changes are approved through proper channels and an *official* updated ATO is uploaded to the fleet's MPSs. Also, since the RMPS will be operating only after some anomalous event caused the AMPS to be shut down, the mission itself may be in jeopardy. If the command center has approved some type of retreat or to abandon the mission and recover the vehicles, we define here that the RMPS will also house a set of pre-certified alternate mission plans to carry out such

changes to the mission. Such alternate mission plans may include changing the role of the mission from surveillance to some other function, such as delivering collected intelligence data to some command center. Or, the mission's objective locations may be redefined to some other, safer locations. The alternate mission may be to retreat to the previous RP, loiter, and await further mission plan updates from the home airbase.

The AMPS will also carry out the ATO, but it is considered to be aggressive in its actions. By that, if additional tasks are seen (such as an unplanned additional surveillance task) that can potentially provide benefit to the mission or overall operations, then the AMPS will attempt to perform these additional tasks. Also, the AMPS may reallocate assets assignments if there is some potential benefit (e.g., one vehicle may have better ISR sensors than another and the AMPS decides to assign that asset to a surveillance task that was originally assigned to a vehicle with less capable sensors). Such runtime alterations of the ATO, however, can come with some risks, such as being detected by enemy forces, collisions with unforeseen objects, running out of fuel reserves, or being unable to meet the timing requirements of the ATO, for example.



**Figure 74. Two Separate Missions Developed to Accomplish the ATO**

### 9.3 Safety at MPS Level

We define safety at the MPS level as simply adhering to all constraints and requirements set forth by the ATO. If some of these constraints/requirements are not met during the course of the mission, then this can have safety implications, both directly to the fleet of UASs (such as flying into unsafe areas) as well as to the overall operations the mission is supporting (such as falling behind on the ATO timing plan, resulting in the fleet's inability to capture critical intelligence data needed for certain troop movements, for example).

We therefore define the set  $S_{Dsafe}$  as the set of states such that the ATO constraints/requirements are met.

### 9.4 Type Safety at the MPS Level

The safety levels that are used to determine the switching condition protocol for an RTA protected MPS block are now formally defined. We will use the capital letter 'M' to denote that these safety definitions are for the MPS level.

**Definition 8.** MPS Type Safety - A point  $x_0$  in the state space  $S$  is:

- **MType I Safe** if that point lies inside  $S_{Dsafe}$  (= adherence to the ATO)
- **MType II with set  $Q_M$  and time  $T_M$  Safe** if all of the following hold:
  - a) Point  $x_0$  is MType I safe,
  - b) Upon switching to the reversionary **MPS**, the state trajectory can converge to at least one point in a *desired* set  $Q_M$  within a given *desired* time  $T_M > 0$ ,
  - c) The state trajectory from the point of switching to the reversionary system to the point of reaching the set  $Q_M$  is entirely contained within the MType I safe region.
- **MType III with Period  $\tau_M$  Safe** if all of the following hold:
  - a) Point  $x_0$  is MType II safe,
  - b) Every possible output of the advanced system for a time period  $\tau_M$  results in a state trajectory entirely contained within the MType II safe region.

Just as with the FMS, the MPS will operate at a much lower bandwidth than the guidance and control loops. Therefore, the period  $\tau_M$  will be relatively large in value (of the order of 10 seconds, for example).

The obvious choice for the desired region  $Q_M$ , would be to regain adherence to the ATO requirements and  $T_M$  would be the time required to regain that state. For example, consider that the AMPS decides to perform an additional surveillance task. However, to do this requires the fleet to fly through a bottleneck, and the time needed to reform the fleet was unanticipated. The MPS RTA determines that continuing to perform the additional surveillance task will cause the mission to fall behind on the ATO timing requirements and the fleets in the vehicle do not have the airspeed capabilities to recover that timing plan. Once this is determined, the MPS RTA monitor shuts down the AMPS and switches to the RMPS. The RMPS's first task is to

command the fleet to get back on the original mission plan defined by the ATO. Once the timing plan is regained, then that defines arriving at the desired region  $Q_M$ .

As a final note, just as was discussed in Subsection 7.3.1, like the FMS, the MPS also operates in future time, planning out the mission over some time horizon. Therefore, the safety scenario just discussed would be discovered in future time, giving the actual RMPS substantial time to recover before actual safety is compromised. In this case the MPS RTA acts as a *software integrity monitor* rather than a safety monitor.

## 9.5 A-G Contracts at MPS Level

Just as with Figure 23, Figure 36, and Figure 44, Figure 75 illustrates how the FMS feedback level is collapsed into the composite *black box* that is certified to be safe and correctly operating at all times (by the compositional reasoning logic). We now develop the A-G contracts at the MPS level for the elements around the bottom feedback loop shown in this figure.

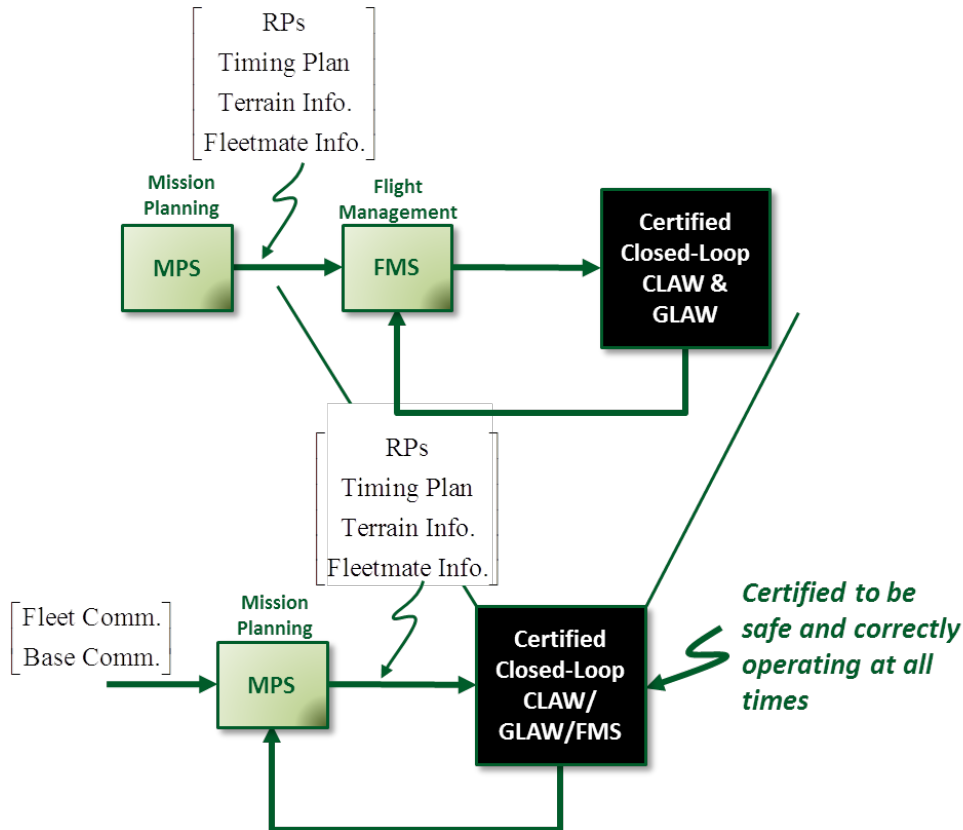


Figure 75. Development of A-G Contracts at the MPS Level

### 9.5.1 Assumptions on the Inputs to the MPS

Inputs into the MPS come from communications with fleetmates and potentially from ground command operators. The following assumption on the inputs to the MPS loop can be listed:

- i. Communication hardware is working correctly (radios for intra- and extra-fleet transmissions working)

- ii. Sensors to detect hazards/obstacles working properly
- iii. Fleetmates are operating correctly; their MPS instantiations are correctly negotiating and executing solutions; fleetmates are correctly communicating their solutions in a timely manner and adhering to the frame rate of the MPS block
- iv. Hazard and other theater/environment information from outside sources is accurate to within specified tolerances
- v. Updates to the ATO are correct in type and format and adhere to all constraints on the mission and fleet.

In summary, all fleet hardware is working properly and any information from outside sources is correct and will not cause unsafe conditions to the mission plan. We assume here that mitigation procedures would be in place to ensure information integrity is maintained throughout the mission.

### **9.5.2 A-G Contracts for Closed-Loop CLAW/GLAW/FMS System**

The assumptions on the inputs to the closed-loop guidance/control/FMS system are:

- i. Positioning of RPs shall not require climb, descent or turning performance beyond vehicle's capabilities
- ii. RPs shall not be spaced so close together that fleet cannot properly follow them
- iii. RPs shall not be placed in hazardous locations (near objects, terrain that can lead to collisions, etc.)
- iv. RPs shall not be placed in restricted airspace or no-fly zones'
- v. Timing plan is physically achievable and does not violate airspeed limitations on the fleet
- vi. Terrain information is correct to within acceptable tolerances
- vii. Fleetmate information is correct to within acceptable tolerances.

The guarantees of the closed-loop guidance/control/FMS system are:

- i. Vehicle is stable in attitude
- ii. Vehicle is stable in translation
- iii. Vehicle is accurately following commanded path (waypoints), remaining within its defined RSV or safety corridor.
- iv. Vehicle is accurately following commanded airspeeds
- v. Stability and path following performance robust to acceptable disturbances and levels of uncertainty
- vi. Management of RSVs is correct and does not violate separation requirements.

In summary, the vehicle is performing correctly and accurately flying to the commanded waypoint locations and the waypoint placements are correct and deconflicted with fleetmates and other obstacles.

### **9.5.3 A-G Contract for the MPS**

The assumptions on the MPS block are:

- i. The assumptions on the input hold, as listed in Subsection 9.5.1
- ii. All A-G contracts for the closed-loop guidance/control/FMS system hold, as listed in Subsection 9.5.2.

The guarantees of the MPS are that it adheres to the ATO constraints and requirements. As long as these guarantees hold throughout the mission, the MPS will successfully complete the mission, as defined by the ATO.

#### **9.5.4 A-G Contract for the MPS RTA System**

For the reversionary MPS, the A-G contracts will be equivalent to the MPS. As defined, this can also be stated as the reversionary MPS guarantees that at least MType I Safety holds for all time (adherence to the ATO constraints/requirements). Additionally, the desired region  $Q_M$  is achievable within time  $T_M$  when the RMPS is first activated by the RTA monitor and switch mechanism. Again, however, this last guarantee may not be necessary since the switch to the RMPS will occur long before safety becomes an issue. There may also be guarantees placed on any contingency plans that the RMPS may command, but this will be specific to the mission design.

For the RTA monitor and switch mechanism block, the main assumptions are that the information input to this block is valid (correct to within some accuracy tolerance) and that it at least starts its operation while the system is within the MType III safety envelope.

As long as the above assumptions hold, the guarantees for the RTA monitor and switch mechanism are that:

- i. Loss of MType III safety is always detected at first occurrence, while the system is in the MType II safe region, and
- ii. The reversionary MPS is activated when loss of MType III safety is detected.

Again, since the above will occur in future time, the guarantees are essentially that the MPS RTA monitor will always detect an incorrect planning solution and switch to the RMPS once this occurs. The last guarantee is that the MPS RTA will always communicate the reversion switch to the other fleetmate vehicles as well as to ground command centers, if applicable.

#### **9.6 MPS RTA Checks and Switching Conditions**

Recall, the general checks the RTA monitor must perform were listed in Subsection 3.4.1. We can now specify these checks for the MPS level.

1. Safety check: At each update to the MPS RTA monitor, it checks that:
  - a) The future system state to determine if it has left the MType III safe region. If so, the advanced MPS is shut down and function is switched to the reversionary MPS. The MType III boundary may be quite complex because it will account for timing requirements and the ability of the fleet to recover back to the ATO plan. The ATO plan itself may be quite complex, with a long list of constraints and requirements.
2. Output/environment check: The MPS RTA monitor checks that the RP locations and timing plan do not violate any constraints imposed on the closed-loop guidance/control/FMS system. Again, this will involve correct placement of the RPs and achievable arrival time requirements, as previously listed.

3. Performance check: The RTA monitor checks that the advanced MPS is achieving its minimum required performance, however defined and measured. Again, this may involve certain ATO requirements.
4. Input/environment check: The RTA monitor also checks that inputs from outside sources do not violate any constraints imposed on the mission. This would be fundamentally checking that outside sources are not attempting to command the fleet to perform any activities that would be detrimental to the fleet, mission or operation as a whole (e.g., flying directly into enemy fire, or into weather conditions the vehicles are incapable of handling).
5. System hardware health check: As with the other loops, assumptions on hardware and information integrity should be checked and mitigation strategies coordinated with an IVHM or RM system.

## **9.7 Additional Considerations at the Mission Planning Level**

### **9.7.1 Building RTA Checks with a Mission Modeling Language**

Given the wide range of mission operations and specific mission considerations, the set of constraints and requirements defined by the ATO may be complex and it may be difficult for the MPS RTA monitor to perform checks for adherence to these constraints/requirements. Since we are addressing far-future systems in which runtime mission planning (and replanning) can be performed entirely by autonomous, intelligent agents, the need for RTA checking at this level is paramount. Compared to current mission planning protocols, in which remote human operators can provide contingency planning under unforeseen events, future fully autonomous systems will require that either all contingency planning be developed prior to the fully autonomous mission, or the onboard RTA systems have the capability to reason about the anomalous events and correctly replan the mission to address those events. Runtime monitoring that accounts for current scenarios, environmental conditions and mission strategies may need to use a combination of universal mission-primitive checks and current mission specifications.

*Safety* at the MPS level is generally a question of mission risk versus reward. Therefore, one proposed main function of the MPS\_RTAs is to continually monitor some measure of risk/reward factor and if this measure becomes greater than some predefined threshold, then the fleet is commanded back to base. In this manner, the RTA does not evaluate or assess any decision made by the AMPS. Instead it simply asks if the current risk is too high to continue the mission. Although risk/reward measures may be more accurately assessed during pre-mission planning, their estimates may become less accurate during run time due to the *fog of war*. In this case, this may not be the best RTA approach.

There may, however, be more direct, particular measures that can provide an assessment of risk or other decision factors for switching to the reversionary system. Yet, these may be highly particular to the current mission. One new concept to consider is that the ground based mission planning software may consist of a modeling environment in which the field commanders tasked with planning out the mission can make use of certain mission primitives, elements or specifications to aid in constructing the mission plan. These may, in part, take the form of a set of logical relations, (such as linear temporal logics, or LTLs). Such examples might be as follows:

- i. At no time during the mission should all vehicles be operating in sector B.

- ii. 80% of the vehicles should complete the first two objectives by time  $t_1$ .
- iii. If only four vehicles arrive at objective No. 4 by time  $t_2$ , activate Plan B...
- iv. Day of mission no-fly zones are defined to be...etc.

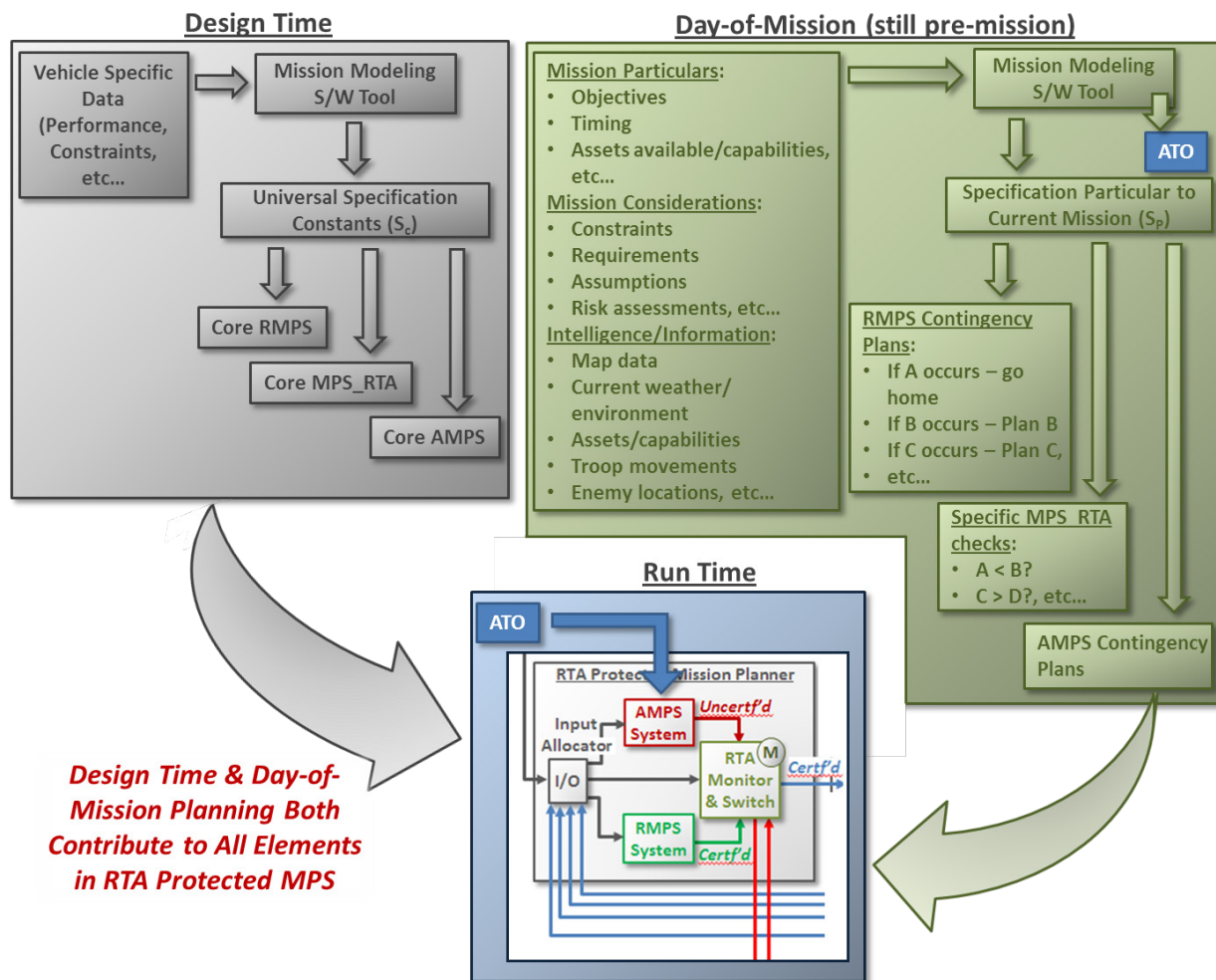
Therefore, the initial planning stages generate not only the initial ATO as defined previously, but also a set of specifications or constraints that must be adhered to during the mission. These specifications would then be continuously checked by the MPS\_RTA. The ground based mission planning software would require some type of mission modeling language (MML) that would give field commanders/planners the ability to rapidly plan out complex missions for fully autonomous fleets during the *day-of-mission* planning and preparations. Because of the urgency in these last planning stages, the approach cannot require complex, labor intensive software recoding of the vehicles' onboard software. However, mission *intangibles* can be complicated to define and change from one mission to the next. The best approach may be to have a graphical MML with easy *pull-down* or *click-and-select* blocks that define mission primitives, with fields for numerical inputs that define the particular values for the current mission. The onboard MPS will then be required to be compatible with this MML environment and can accept these special inputs as part of the mission plan. Last, this MML environment should have *certification authority* in that no mission plans generated by the software have any inconsistencies that would put the mission at risk of failure. This would be accomplished by automatic validation checking of the inputs made by the users of the software package.

In summary, we consider that the RMPS and the MPS\_RTA are comprised of two main parts:

1. a defined *core* for the RMPS and a defined *core* for the MPS\_RTA that are reversionary plans and specifications to be checked that are universal to all missions, developed at design time, and
2. a *day-of-mission* set of reversionary contingency plans that define the rest of the RMPS and a *day-of-mission* set of specifications or constraints to be checked by the MPS\_RTA.

That is, we propose that the field commanders/mission planners have the ability to *construct* parts of the RMPS and the MPS\_RTA during day-of-mission planning. At runtime, these augmented elements are loaded onto the core platforms on each vehicle in the fleet. Further, during the day-of-mission planning process, contingency plans and other elements particular to the mission may also be constructed for the AMPS as well, depending on design protocols and capabilities of the MML software tool. Figure 76 presents this protocol, indicating that the RTA protected MPS consists of both design time and day-of-mission planning constructs. It is evident that the advanced and reversionary elements of the MPS as well as the RTA monitor will most likely be highly integrated at the mission planning level. In summary, RTA system constructs will be an integral part of the battle rhythm and air tasking cycle, presented in Figure 69 and Figure 70.





**Figure 76. RTA Protected MPS Consists of Both Design Time and Day-of-Mission Planning Constructs**

### 9.7.2 Risk/Reward Management

Planning missions will involve risk and reward assessments, both in the pre-mission planning stages and in real time during mission operations. For example, UAS survival may or may not take precedence over completing objectives, depending on the relative importance of the mission and cost of equipment. However, safety of the fleet will typically take precedence over safety of any one UAS. And, logically, no UAS should interfere with the progress of the overall mission and should self-sacrifice or return to base if, for example, battle damage causes the UAS to be unable to precisely control its position, thus increasing the likelihood of collision with another in the fleet. We assume the mission starts with notional values for priorities for mission objectives, risks in completing mission segments (e.g., flight near mountainous terrain, or enemy anti-aircraft batteries) and value assessments for each UAS in the fleet (e.g., high value assets are spared from the riskiest parts of the mission). Zones or certain locations to be avoided may be assigned different risk values depending on the dangers they present. Airborne intruders will also be assigned a risk if target identification can be performed. For example, if it is known that the intruder is a friendly onboard piloted aircraft, then it should be assigned a high risk level (i.e.

the UASs must not get close to the piloted aircraft). If the intruder is known to be a piloted enemy aircraft, that too will carry a high risk value for reasons of potential interception or detection. However, if the intruder is known to be an unaware UAS, then the risk level may be lower. Mission priorities, equipment value assessments, and mission segment risks can change values over the course of the mission due to battle damage, unfolding/changing scenarios, newly acquired intelligence data, and changes in mission objectives.

RTA systems at the MPS level will most likely use risk/reward measures in some of the decision making. How much risk versus mission completion the RTA system is willing to accept should be defined by a *user defined strategy or mission profile* (i.e., defined as part of an input data file or I-load). We propose this approach as a *recommended practice* for designing/developing an RTA system. The advantage here is that the same RTA system (same code) can be used with day-of-mission inputs defining its characteristics ranging from a *nervous personality* to a *relaxed personality* depending on the application at hand.

### **9.7.3 Nominal Contingency Planning and the Challenge to RTA Monitoring**

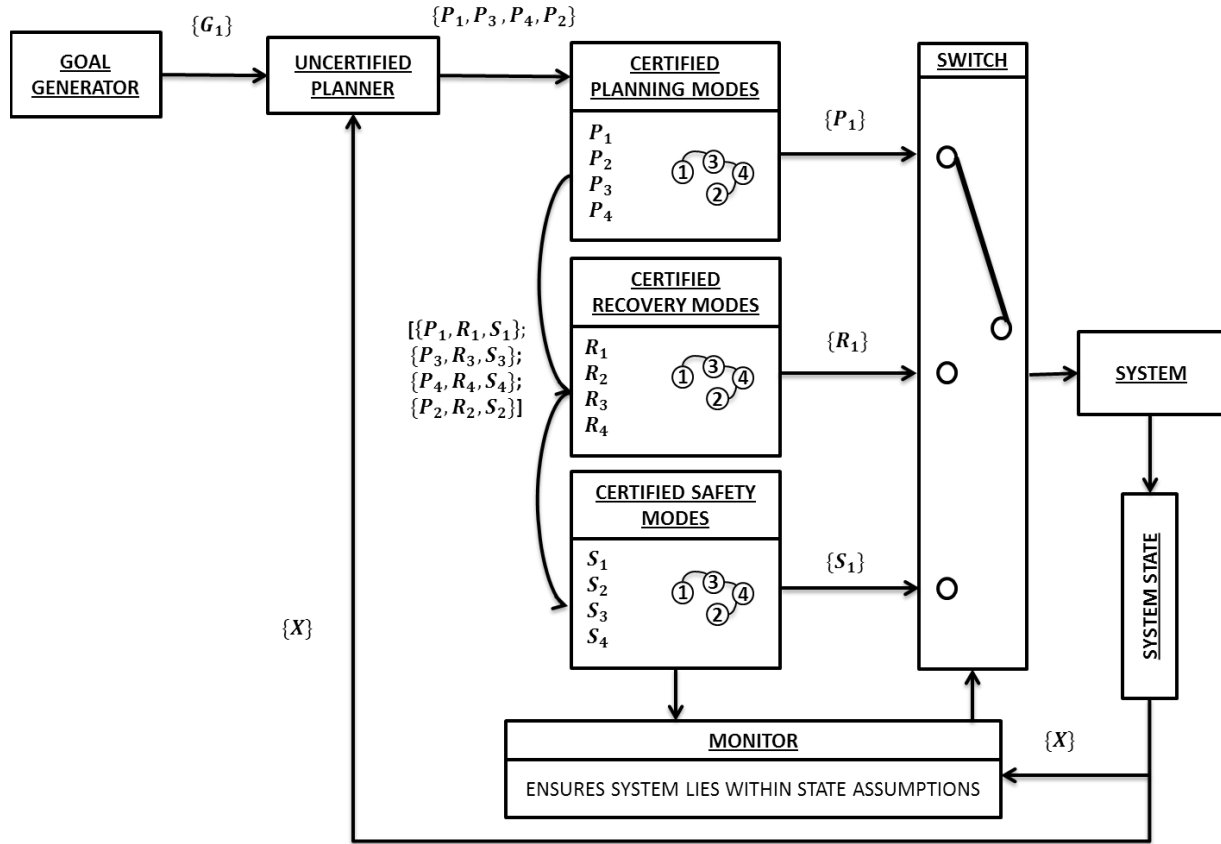
It is expected that the advanced system would be constructed to be resilient and adaptable to anomalous events. Such events should not always trigger reversion to the RMPS. Contingency mission plans may not necessarily be defined to go back to the home airbase. Abandoning the mission may not always be an option. For example, if UASs are escorting a ground VIP vehicle, the VIP vehicle cannot be left in the field and the UASs must continue to escort and scout out pathways no matter what unforeseen events are encountered. One objective in the VIP escort system application is for the VIP ground vehicle to keep moving at all times to reduce its vulnerability. The role of the UAS fleet is to fly ahead of the VIP and scout out road segments for possible threats. They then fly back to a designated rendezvous point, share intelligence, and deliver directions to the VIP (e.g., “turn left at the next intersection”). However, if all UASs do not arrive back at the rendezvous point in time before the VIP enters the intersection, the vehicle must turn around and *retreat* back the way it came so that it keeps moving. This, therefore, is a built in contingency plan, but is not necessarily a failure of the AMPS, so reversion to the RMPS is not warranted at this point (in fact, it may be expected that the VIP will turn around several times over the course of its journey to its destination). Other missions may also require completion, such as a UAS fleet delivering supplies or vital battle information. Failure to complete the mission may put ground troops in a hazardous situation. In general, there may be a *library* of contingency plans that can be drawn from and these may be a natural part of the AMPS.

This presents a challenge to the construction of the RTA system at this planning level. Since a number of contingency plans may actually be a part of the original AMPS Plan, when such plans are activated differentiating expected contingencies from contingencies due to errors in the advance system can be difficult. The RTA checks must account for this and not unduly command reversion to the RMPS.

### **9.7.4 AFRL Proposed General RTA Modal Architecture**

At the MPS level, much of the operational protocol may be characterized by managing modes of operation, such as surveillance, refueling, attack, retreat, support, return to base, etc. There may be hierarchical mode structures and certain define mode stitching requirements or constraints

(e.g., mode B cannot start until mode A completes, or mode C must follow mode B, etc.). AFRL proposed a general RTA modal framework and this is shown in Figure 77. The idea behind this framework was generally drawn from [Majumdar 2013], [Tobenkin 2011] which focus on trajectory planning systems, such as robotic motion primitive sequences. A generalized modal framework may involve planning modes, operational modes, etc.



**Figure 77. AFRL Proposed General RTA Modal Architecture**

The main elements of this framework are as follows:

1. Goal Generator: is the set of instructions provided to the autonomy. The goal would specify the desired outcome  $\{G_1\}$ . The planner would receive  $\{G_1\}$  and the current state  $\{X\}$ .
2. Uncertified Planner: The uncertified planner compares the current state  $\{X\}$  with the desired goal condition  $\{G_1\}$ . The planner applies *uncertified* methods to pick an appropriate response based on unknown / not previously encountered environment variables presented in  $\{X\}$ . The response is translated into a sequence of pre-certified modes, each mode having an appropriate response to a portion of the Goal that lies in the domain  $\mathbb{R}^{\{P_n, R_n, S_n\}}$  which exists in  $\mathbb{R}^{\{X\}}$ , stitched together to accomplish the desired goal.
3. Certified Mode Library: The certified mode library contains sets of hybrid modes  $\{P_n, R_n, S_n\}$  containing three distinct components; 1) a *planning mode*  $\{P_n\}$ , 2) *recovery mode*  $\{R_n\}$ , and 3) a *safety mode*  $\{S_n\}$ . All sets are verified *a priori*, providing a guarantee of

safe operation within the region  $\mathbb{R}^{P_n, R_n, S_n}$ . Note that the recovery mode here is analogous to the transition system and the safety mode here is analogous to the baseline system in the RTA protected framework.

4. System: The system that is being protected from undiscovered software faults in the uncertified planner.
5. Monitor: The monitor checks that the assumptions provided in the sets are not violated given the current state. It does not consider the current goal but predicts into the future to determine if all certified planning modes are reachable. If this is not the case, it switches to the sequence of recovery modes, which take the system to the sequence of safety modes.

Because modes will be operating in an interactive manner with other modes or subsystems, or in some hierarchical manner with parent and child modes, they should be defined to have a specific structure describing their properties for analysis and design validation. Table 6 presents some proposed requirements for defined mode properties. We do not consider this an exhaustive list and further information may be needed to fully define the mode.

**Table 6. Proposed Required Mode Property Specifications**

| <b>Mode Properties</b>                                  | <b>Description</b>                                                                                                                                                                                                                                       |
|---------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Objective(s)                                            | The goal, mission or purpose of the mode                                                                                                                                                                                                                 |
| Influences                                              | Other system elements that influence the operation of the mode                                                                                                                                                                                           |
| Assumptions/<br>Assertions                              | Assumptions and rules for when and under what circumstances the mode can successfully operate                                                                                                                                                            |
| Certified<br>Operating<br>Envelope                      | Upper and lower bounds on all critical parameters and states that have been tested at design time V&V                                                                                                                                                    |
| Boundary<br>Certificate<br>Requirements                 | Safe and unsafe regions of operation for the mode                                                                                                                                                                                                        |
| I/O Structure or<br>Interface Control<br>Document (ICD) | Information required for the mode to operate and its origin<br>Data generated by the mode during its operation and its destination<br>Note: all data quality and characteristics, and required bandwidth or update rate, etc. should be clearly defined. |
| Associated<br>Dynamics                                  | If the mode is associated with particular dynamics or other physical properties, the key aspects of these dynamics that can influence the mode's behavior need to be described.                                                                          |
| Constraints                                             | Constraints and limitations of the mode.                                                                                                                                                                                                                 |
| Performance<br>Guarantees                               | Performance or operational function the mode is guaranteed to achieve – given all assumptions are valid                                                                                                                                                  |

The required RTA monitoring should perform the following runtime checks.

### **Mode Interaction Level RTA Checks**

1. Stitching of modes (or operational plan) remains feasible so that the system can successfully transition from one mode or operation to next.
2. The final objective is still achievable under the current conditions and mode/operation plans, i.e., the mission is not diverting off to the wrong path.

If either of these cases is observed, the RTA system must make sure an alternate safety plan or re-planning strategy is available that can achieve some type of recovery, such as getting back *on course* to achieve the original goals, or to revert to the certified baseline operation. Note here that the baseline plan or operation may be to return or retreat back to base, perform a holding or loitering plan, etc. Therefore, there will be the need for some predictive capability in the RTA system because it needs to continually “look ahead” to see if what is planned will be feasible.

### **Mode Level RTA Checks**

Here, the RTA system should check whether any mode properties are being violated. That is:

1. Mode objectives can still be achieved.
2. Assumptions/assertions are not being violated.
3. Operation is within permissible envelope.
4. I/O constraints are not being violated.
5. Governing dynamics are feasible.
6. Other constraints are not being violated.

## **9.8 Proposed Experimental Investigations**

Although no experimental simulations of RTA protection at the MPS level were completed during the course of this project, a number of candidate experiments were developed *on paper*. These are presented here as a matter of record and may prove useful starting points in follow-on programs.

**Heterogeneous and Homogeneous Fleets:** Fleets of different vehicles with different capabilities may pose unexpected problems and unforeseen emergent behavior to the MPSs performing continuous intrafleet communication and coordination. Although we presented a system that has RTA protection at all feedback levels, this may not be the case for all vehicles in the fleet. For example, some vehicles may have advanced inner-loop controllers requiring RTA protection, whereas others may have fully certified, simpler inner-loop controllers with no RTA protection required. Likewise, the outer-loop guidance systems as well as the FMSs may be different for each vehicle, with some having RTA protection, some not requiring RTA systems. Therefore, if total system (fleet) reversion is necessary at the MPS level, coordination among all vehicles in the fleet will not be straightforward and may pose coordination and timing issues. Such issues may also arise for individual platforms that revert, causing the entire fleet to unnecessarily revert. That is, can failure of one vehicle’s inner-loop control system and subsequent RTA control mode reversion cause a major false alarm that ends the entire mission?

Related to this, notionally all vehicles in the fleet should be using the same MPS (made up of the AMPS, RMPS and MPS\_RTAs) so that a consensus regarding the fleet’s mission plan is rapidly achieved. However, due to the *fog of war* this may not always be the case and one vehicle may be using an out-of-date version of the AMPS software, for example. Or, half the fleet is using

one version of the code associated with the RTA system, and the other half is using an updated version. Such inconsistencies may lead to nonconvergence of the fleet's mission plan, or unforeseen behavior in the reversionary process.

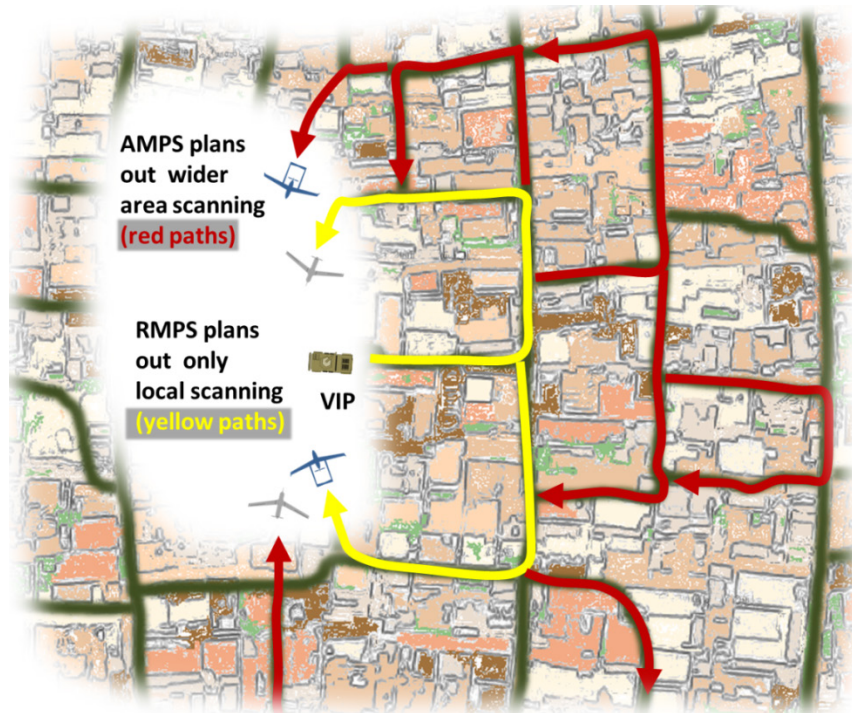
Even for homogeneous fleets, it may be that not all vehicles will be in close proximity with the rest of the fleet at all times, and therefore sensor information may be disparate, causing different MPS solutions to the fleet mission plan. However, for homogeneous fleets, common mode failures in the RTA systems could lead to complete fleet reversion under false alarm, or worse, failure to identify the common source of error in the AMPS software, leading to hazardous conditions during the mission.

Downstream subsystems within a particular vehicle in the fleet may experience problems due to unforeseen events, such as battle damage, or nonconvergence of the AFMS, AGLAW or ACLAW systems. Any one of these downstream subsystems may therefore change modes to their respective reversionary systems, or because of physical damage to the vehicle itself, the UAS now has different dynamic characteristics or degraded performance capabilities. This then leads to unplanned heterogeneity in the fleet that must be dealt with by the MPS\_RTA system.

**VIP Escort Timing Issues:** Again, one example in which fleet reversion does not imply ending the mission is the VIP escort problem. A candidate scenario is as follows. Consider that the AMPS's function is to send its entire fleet out to scan a wider area of road segments and intersections using an advanced target recognition software package. This requires a longer time for scanning and a failure in the advanced system may be that the target recognition algorithm takes longer to process than expected. This results in the UASs not being able to arrive at the designated RP before or at the same time as the VIP. Hence, the VIP has to turn around in order to keep moving. This, of course, causes delays in achieving the final destination. Or, there may simply be a constraint that no data gathered can be older than a certain time threshold, and the advanced system is not able to consistently meet that threshold, thus requiring rescanning of the roadways.

The RMPS's role would be to simply send subsets of the fleet to scout out a smaller, more immediate area. Visual feed of the roadways would be sent back to the VIP operator to assess whether dangerous conditions are present. Although slower in progress than the intended advanced system's performance, the reversionary system would have fewer instances in which the VIP vehicle would be required to turn around.

Figure 78 illustrates this concept, showing an aerial view of an urban setting in which the AMPS plans out a much wider scanning area, involving two to three city blocks in several directions using the entire fleet. Conversely, the RMPS only scans the most immediate blocks nearest the VIP vehicle with a smaller number of UAS assets.



**Figure 78. AMPS and RMPS Scanning Protocols for the VIP Mission**

**Risk Monitoring:** At the MPS level, *self-survival* and *self-sacrifice* logic or analysis may take place. Depending on the negotiations between the AMPSs for all vehicles, there will need to be re-planning and re-negotiations until all assigned roles are decided. Much of the decision making will rely on relative importance of the mission over the importance of bringing back the equipment. More important missions may accept the risk of losing some of the vehicles. Or, high-value UASs may take on the less risky assignments, for example. Or, a UAS with significant battle damage may now become a low value asset and, if capable, take on a more risky assignment. The RTA system may be utilized to manage the relative risk/reward measures and to provide basic checks within all the decision making processes that the proper risks are being taken.

**Decentralized Framework and Coordinating Sub-Fleets:** The decentralized nature of the MPS may require RTA monitoring. The fleet may start out all within communication range and therefore start with the same AMPS plan solution. But, as the mission evolves, the fleet may split into several sub-fleets that are not in communication with each other. In this case, each sub-fleet has its own AMPS plan solution. Also, vehicles may come into or out of a sub-fleet, changing the solution. With communication delays and time for negotiations and re-planning, at any given time it cannot be assumed that all vehicles within a sub-fleet will have the same AMPS plan solution. As they fly along, they may tend toward the same solution as all the vehicles negotiate their individual roles. Some RTA checks may be needed because of this to ensure solutions that result in some type of conflict (be it a collision course, or confusion about assigned roles, etc.) are de-conflicted in some manner to ensure safety of the fleet.

## 10 Safety Case Argument for an RTA Protected System

### 10.1 General Considerations

Developing safety case arguments for aircraft systems that include RTA protected systems will require arguments for safety for the complete aircraft and all its subsystems. In this chapter we provide arguments that focus mainly on the protection and mitigation processes provided by RTA systems. However, we recognize that a complete safety case must ultimately include arguments for safe operation of all systems and subsystems that interact with the RTA protected elements to ensure only intended functionality exists and no unwanted functions, interactions or processes occur due to the inclusion of RTA protection.

Several challenges exist in order to make a plausible safety argument for RTA systems, such as those noted in this representative (and incomplete) list:

1. RTA has validated requirements as proven by the safety assessment at the aircraft level.
2. RTA is independent and not affected by the faults of the monitored software/functions.
3. RTA itself does not interfere with the nonfunctional attributes such as availability, integrity, safety, performance, etc. of the flight critical systems.
4. RTA does not introduce any new hazards because of its own function – scheduling problems, live-lock, deadlock, lost links and other conditions that can lead to *paralysis* of the RTA function.
5. There should not be any single point of failure.
6. The trigger point at which RTA switches to the reversionary software does not introduce any hazards to continued safe flight.
7. This trigger itself should be reliable and correct at all times.
8. Construction of RTA must be diligent and correctness must be demonstrable.
9. Use of RTA may necessitate that anomalous states be retrievable for post-incident/accident investigation.

There are other conditions that must be noted along with the interaction between the various system components and their assurance. Again, the safety case must be at the aircraft system level with RTA as one of its components.

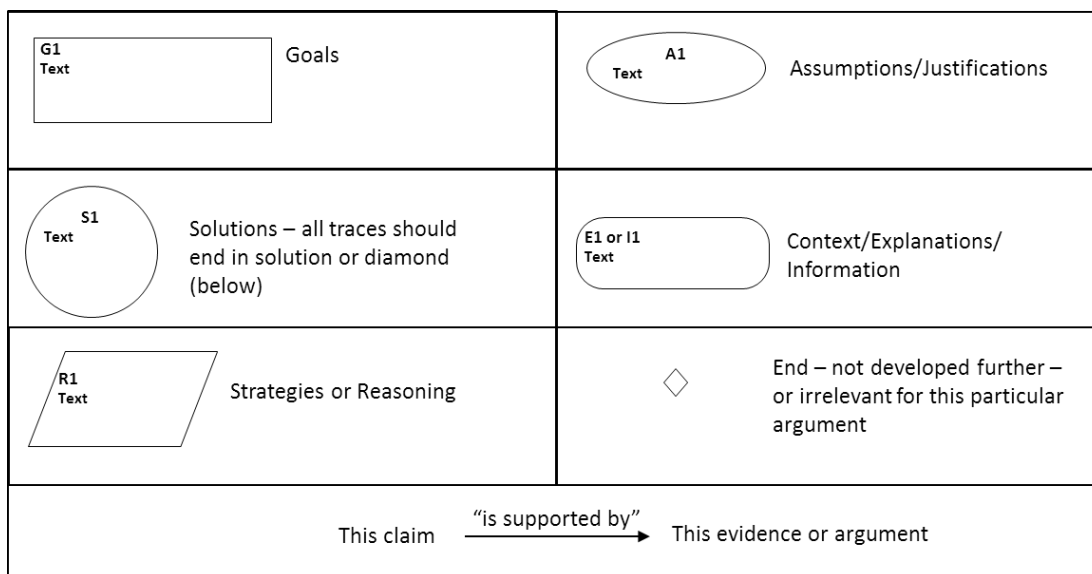
Use of a safety case to make the argument that RTA protection provides an *equivalent level of safety* as a DTA system must be backed up by robust engineering practices that can demonstrate the above conditions. A safety case is not a substitute for required engineering; rather a safety case can be used to organize the arguments to make a case. Although a safety case allows for various different suitable engineering methods to be used, the applicant and the regulator have the burden of making sure that the method used is appropriate to the problem at hand. A natural result of the development of a safety case is that the RTA developer has the responsibility to convince the regulator that all proper considerations have been appropriately addressed in the engineering of the RTA system.

### 10.2 GSN Diagrams used to Construct Safety Cases

The tools provided by the goal structuring notation (GSN) approach [GSN 2011], [Despotou 2010] are fundamental to constructing safety cases. Figure 79 presents the main safety argument tools that are used in the GSN approach. Fundamentally, the different elements of an argument



are represented by different shapes, and these shapes are put together in a logical flow diagram. This approach has gained recent interest as a way to logically formulate arguments for safety of a system that support the standardized process approaches given in ARP4754A and DO-178C.



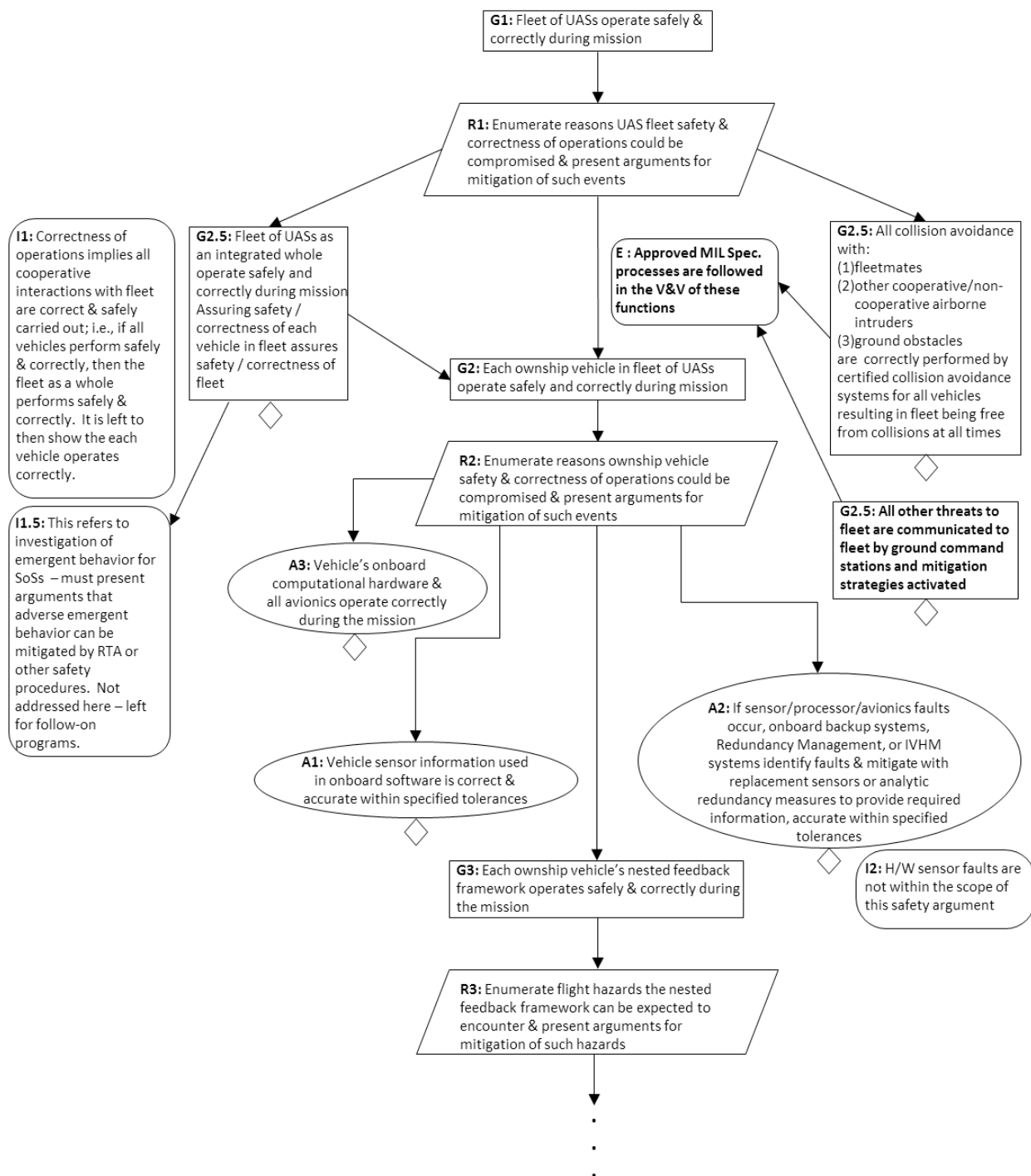
**Figure 79. Safety Argument Tools Using Goal Structuring Notation**

Representation of arguments in GSN provides several advantages including provision of:

- A uniform, standardized notation permitting consistent communication of arguments.
- A well-defined and accessible structure for the overall argument that supports detailed scrutiny and analysis of the argument.
- Complete explicit representation of essential content.
- A structure within which various forms of quantification can be described and analyzed with dependencies documented explicitly in the graph structure of the goal decomposition.

### 10.3 Safety Case for Challenge Problem using GSN Diagrams

This section presents a preliminary safety case argument for the RTA protected system of the challenge problem using GSN diagrams. Figure 80 presents the top level safety argument for the fleet as a whole. This figure states that we are not addressing complex interactions in fleets (such as emergent behaviors) and it is sufficient to prove that if all individual vehicles are operating safely and correctly, then the fleet as a whole is also operating safely and correctly. Further, we are not addressing hardware or sensor failures here, so it is assumed all equipment is operating correctly. With this, we next present safety arguments for an individual vehicle, one feedback level at a time.



**Figure 80. Top Level Safety Argument for Fleet of UASs**

Figure 81 presents the safety argument pattern template that was used for each feedback level. It can be seen that we relied substantially on the A-G contracts and compositional reasoning logic we have already developed to formulate this safety argument pattern. Fundamentally, we start each argument with the assumptions that all upstream and/or downstream feedback levels are operating safely and correctly. We then provide the safety argument for that particular feedback

level and show how RTA monitoring and failure mitigation with the reversionary system supports the safety argument even with the uncertified advanced element included in the operation of that level. Those elements that can be certified at design time are proven safe by the accepted certification processes. We note this in the arguments as coming from MIL Spec. standards since this is for an Air Force application. However, for commercial applications these safety artifacts would rely on the processes defined in ARP4754A and DO-178C. The safety argument concludes with the goal that that level is safe and correctly operating. This goal is then used as the top level assumption in the next downstream feedback level.

Using this safety case pattern, Figure 82 presents the safety argument at the MPS level, and Figure 83 presents this argument for an example case study. In summary, the AMPS delivers a mission plan that includes additional surveillance tasks. These additional tasks, if carried out, will cause the ATO Timing Plan to not be met due to airspeed limitations of the vehicles in the fleet. If the timing plan cannot be maintained, air traffic management airspace coordination and planning will be compromised and timely delivery of collected intelligence will be at risk. This risk is mitigated by the RMPS taking over operations and returning mission progress to the original ATO plan.

Figure 84 presents the safety argument at the FMS level, and Figure 85 presents the safety argument at the FMS level for a particular case study. In summary, the AFMS delivers an aggressive plan of RSV placements that violate separation requirements through a bottleneck, encountered because of additional surveillance tasks planned out from the AMPS. Separation requirements are violated in order to keep up with the ATO Timing Plan. The FMS RTA detects the separation requirement violation, shuts down the AFMS to mitigate the event, informs the MPS RTA system of its action and waits for further instructions. The RFMS deconflicts the RSVs of fleet, but this causes ATO Timing Plan to not be met. The MPS RTA then reverts to RMPS and follows the original mission plan of the ATO. This case highlights how a series of cascading failures made by the advanced systems at different levels can all be mitigated from one level to the next by their RTA protection elements.

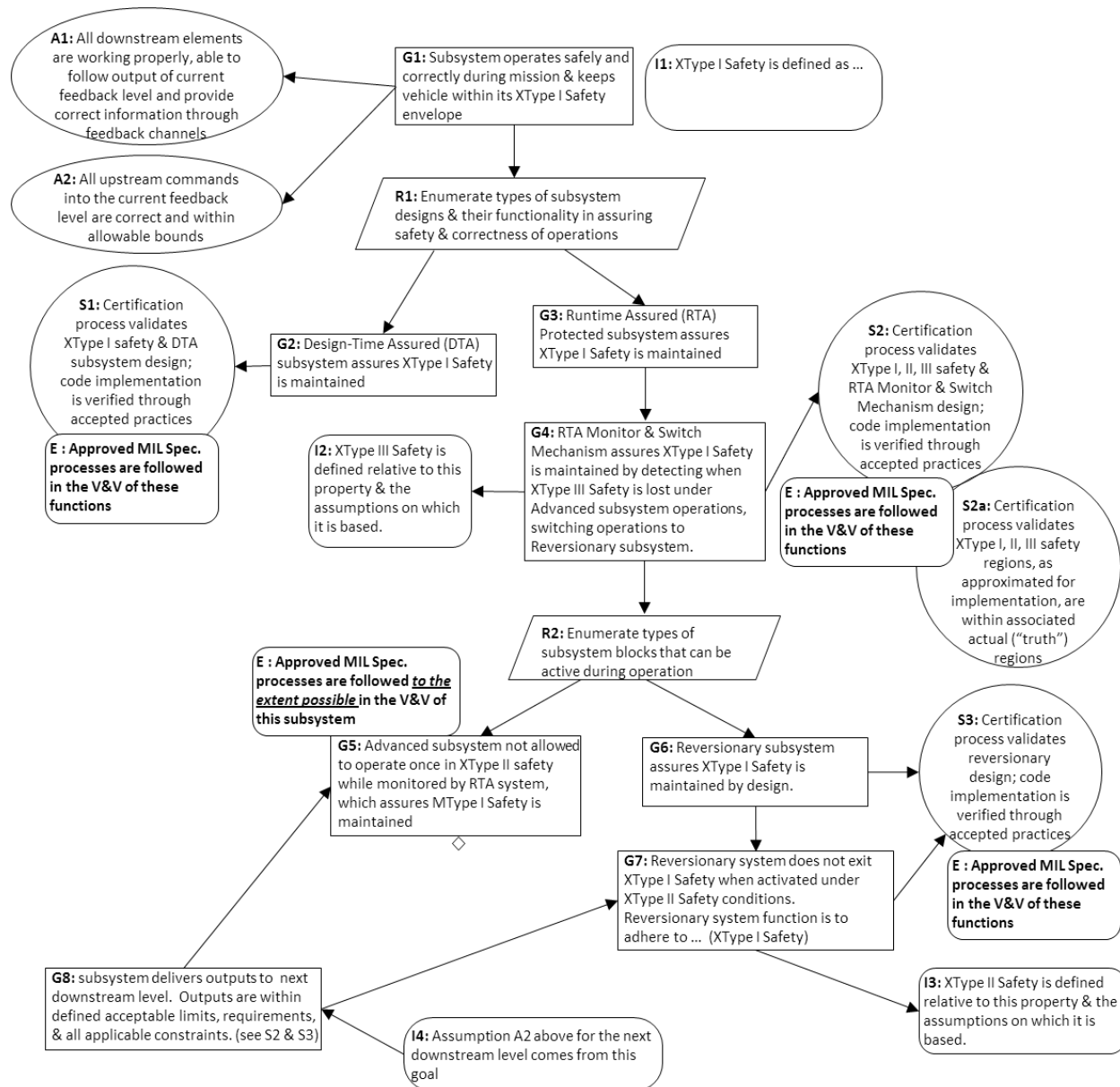
Note that a similar safety case study can be developed for the example of fuel reserve depletion due to faulty AFMS placement of waypoints from failures in the advanced RRT algorithm, as covered in Subsection 7.6.1.

Next, Figure 86 presents the safety case argument for the GLAW feedback level and Figure 87 presents a particular case study at this level. Here, due to a fault in the AGLAW, the combined crosstrack and bearing errors grow to point at which a switch to the RGLAW must occur to avoid an overshoot in crosstrack that goes beyond the RSV boundary (GType I safety boundary).

Last, Figure 88 presents the safety case argument for the CLAW feedback level. In a similar manner, case studies can be included in this pattern, such as the examples on a mismanaged wing morph or the error in the parameter ID algorithm used in the advanced controller, as presented in Subsections 5.6.1 and 5.6.2.

To further mature this overall safety case argument, additional details should be added on checking that performance requirements are being monitored and maintained by the RTA protection elements, as well as the other checks the RTA monitors perform. Although touched

on in the top level safety case, additional detail should be added regarding how a certified collision avoidance system protects the fleet from unforeseen intruders/obstacles using the framework presented in Figure 61. Last, although we stated in the top level argument that we did not further develop considerations regarding hardware failures, this aspect should be expanded and details added involving how the RTA systems interact with RM/IVHM systems and adaptive controllers to mitigate these failures.



**Figure 81. Safety Argument Pattern Template for Feedback Level**

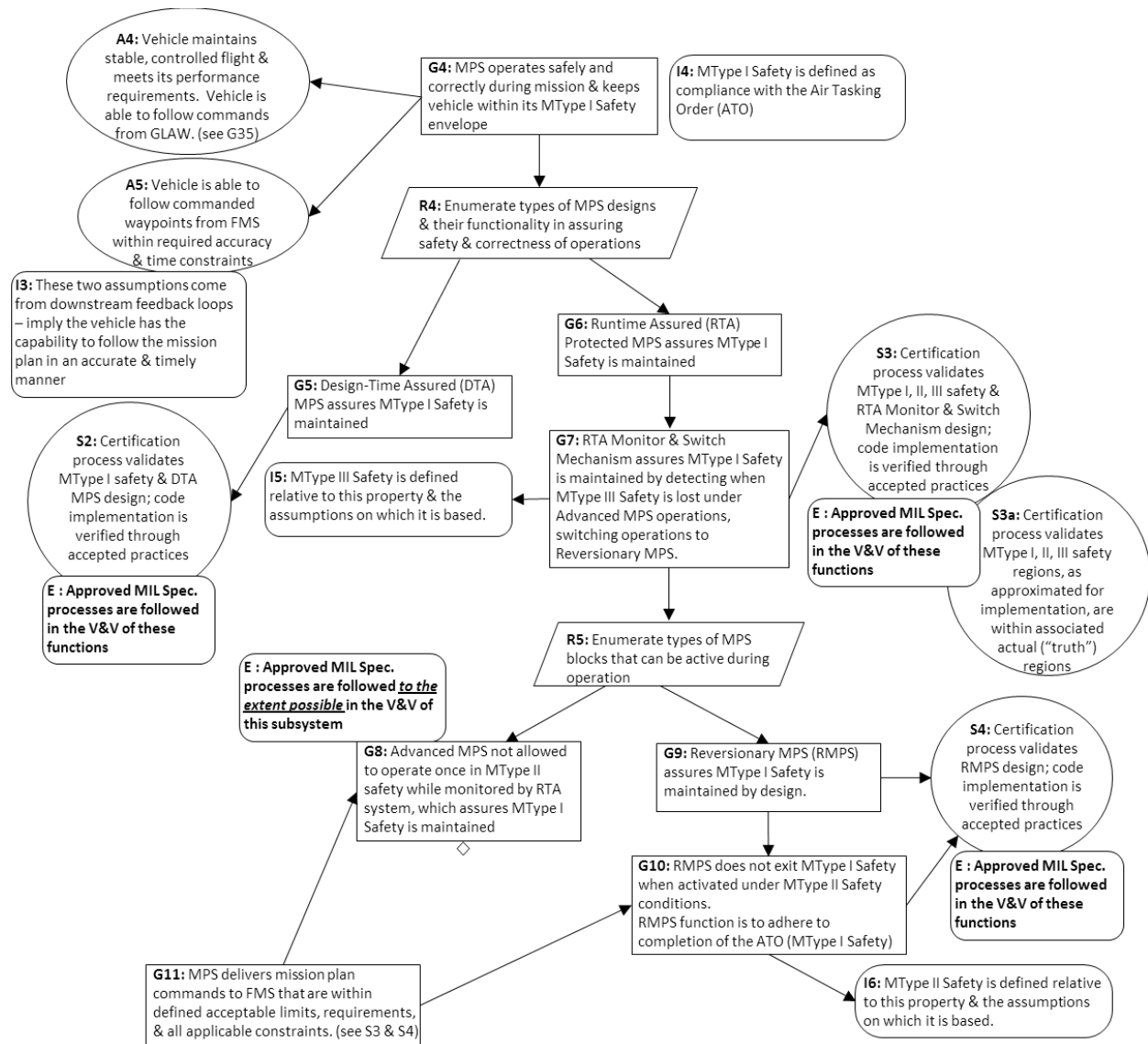
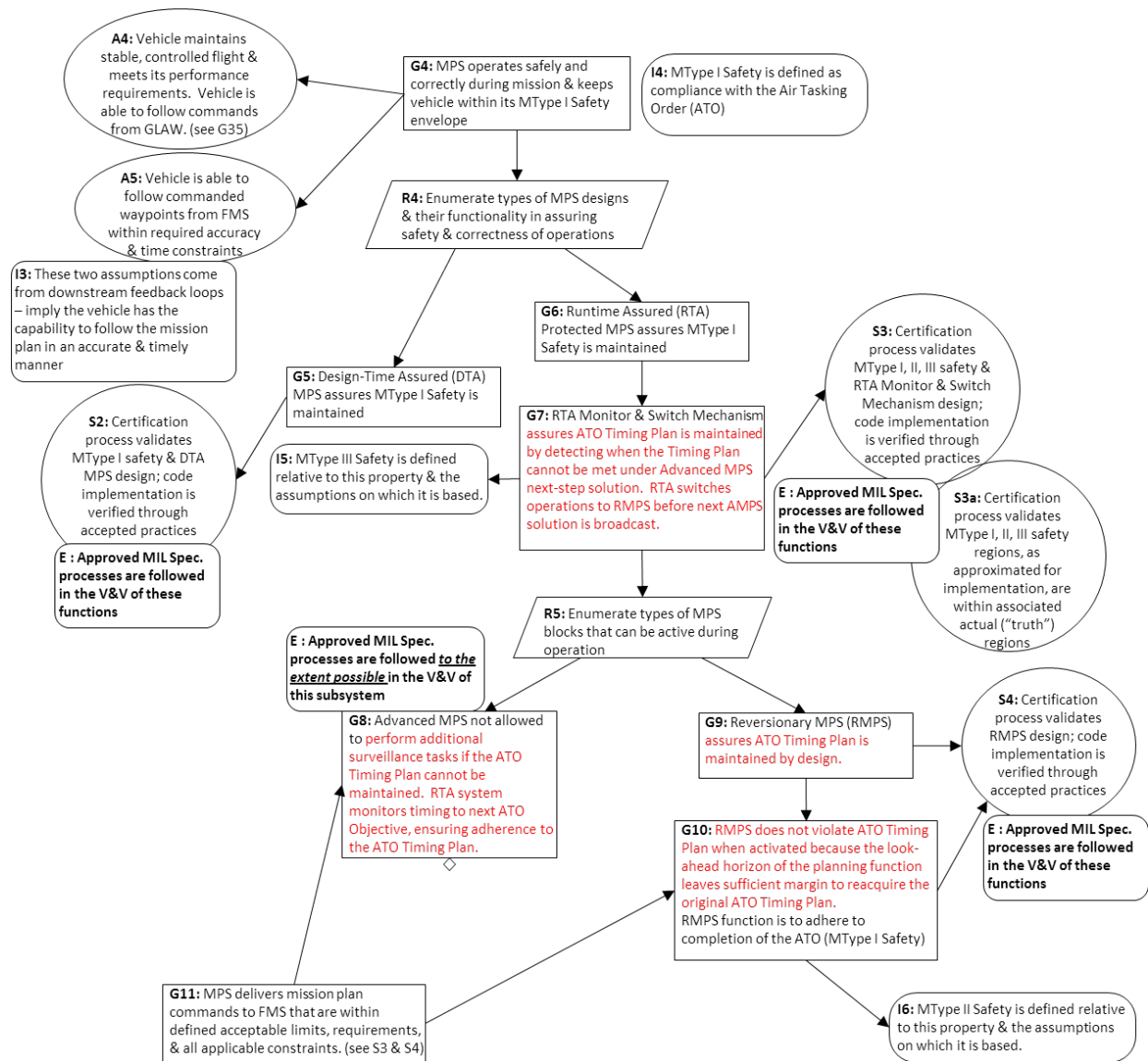
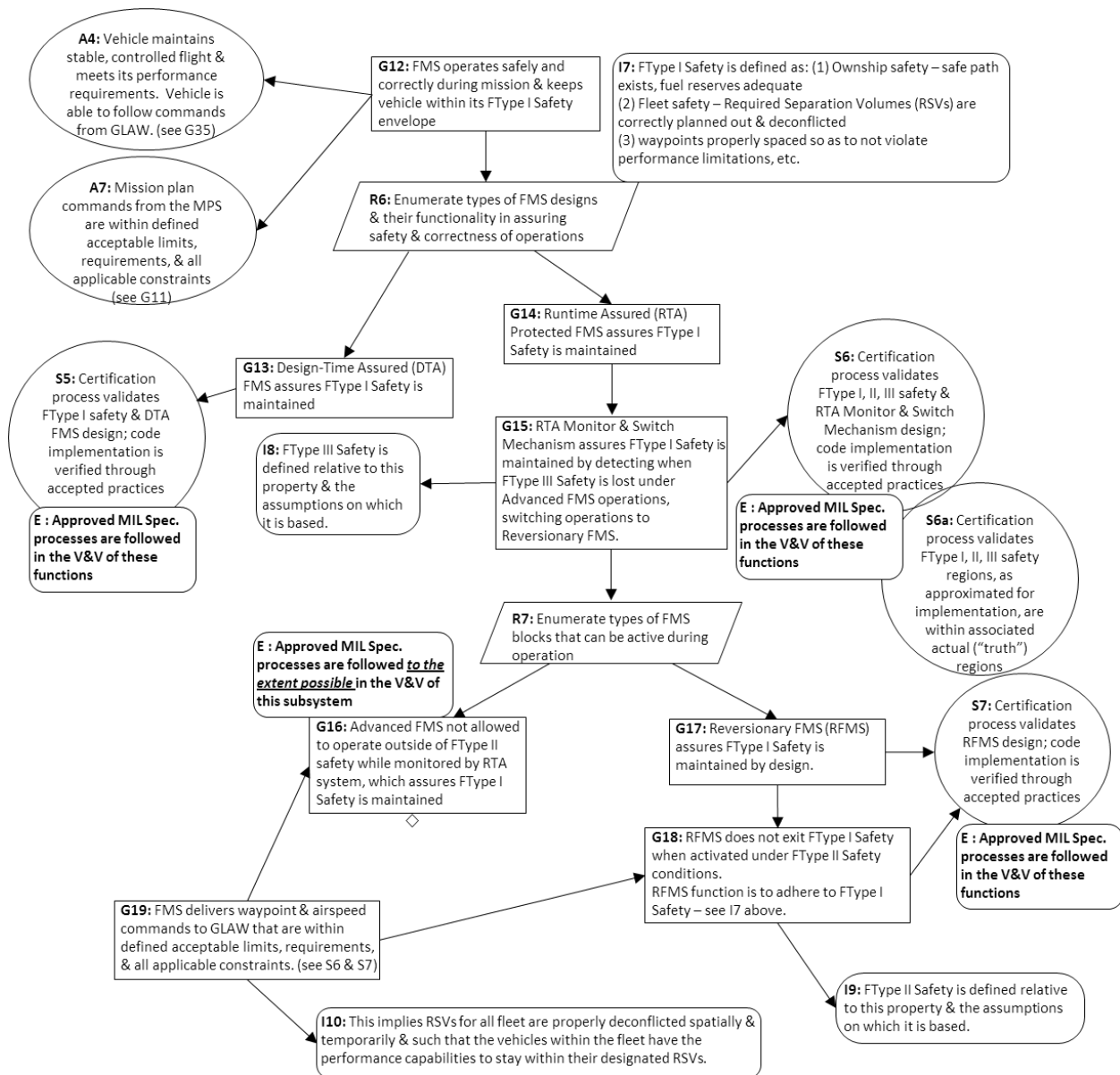


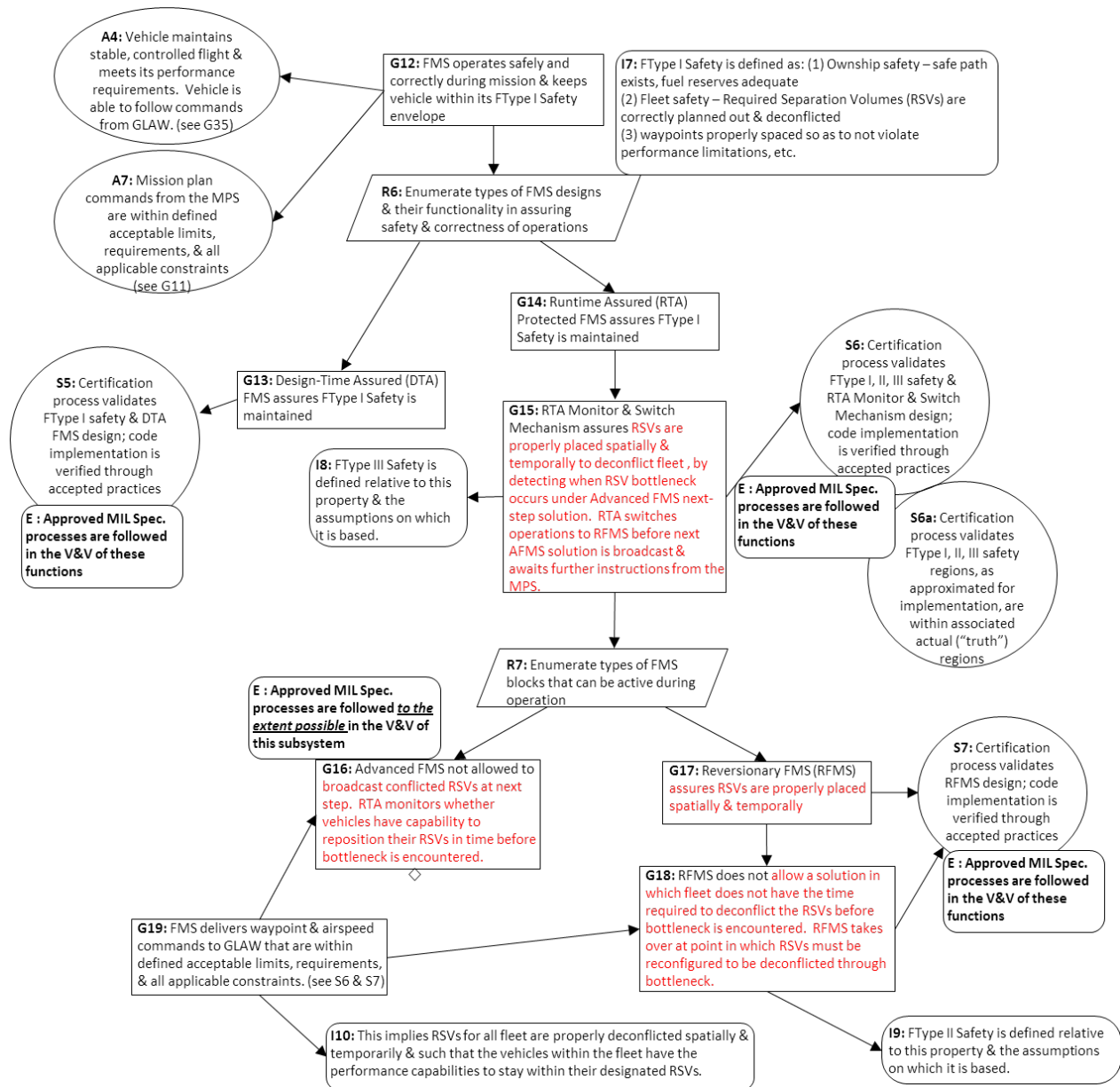
Figure 82. Safety Argument for MPS Feedback Level



**Figure 83. Safety Argument for MPS Feedback Level – Example Case Study**

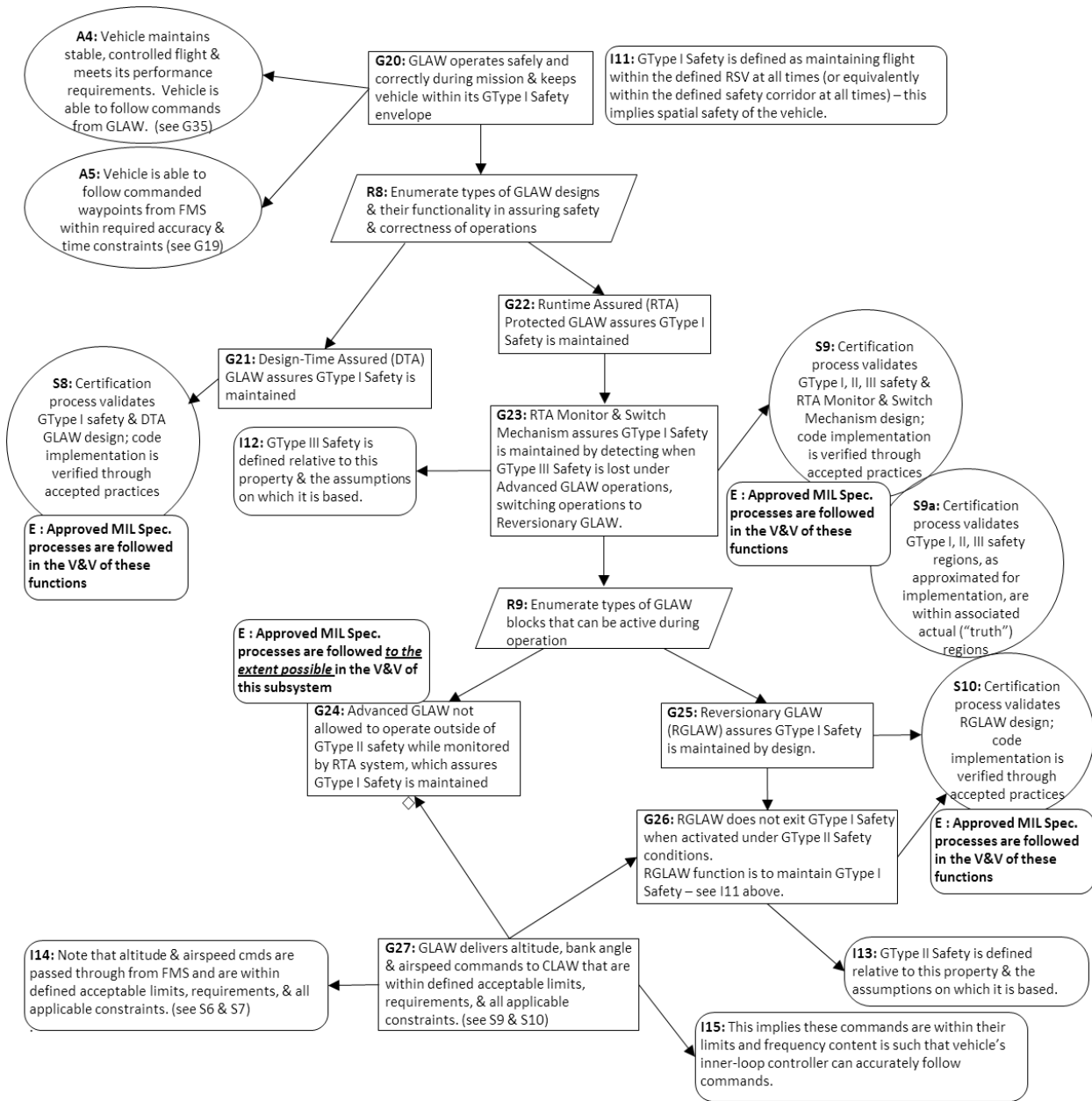


**Figure 84. Safety Argument for FMS Feedback Level**



**Figure 85. Safety Argument for FMS Feedback Level – Example Case Study**





**Figure 86. Safety Argument for GLAW Feedback Level**

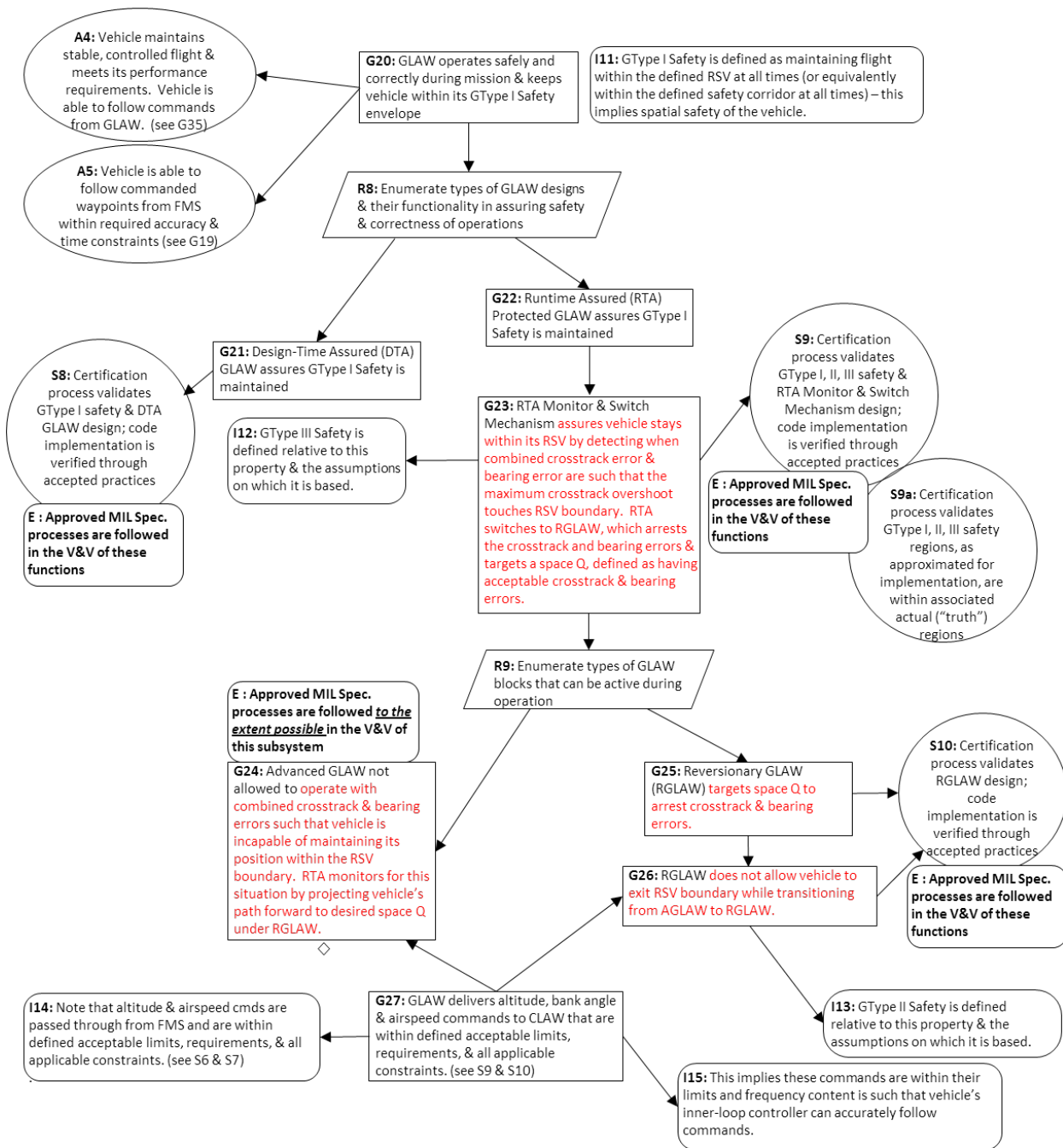


Figure 87. Safety Argument for GLAW Feedback Level – Example Case Study

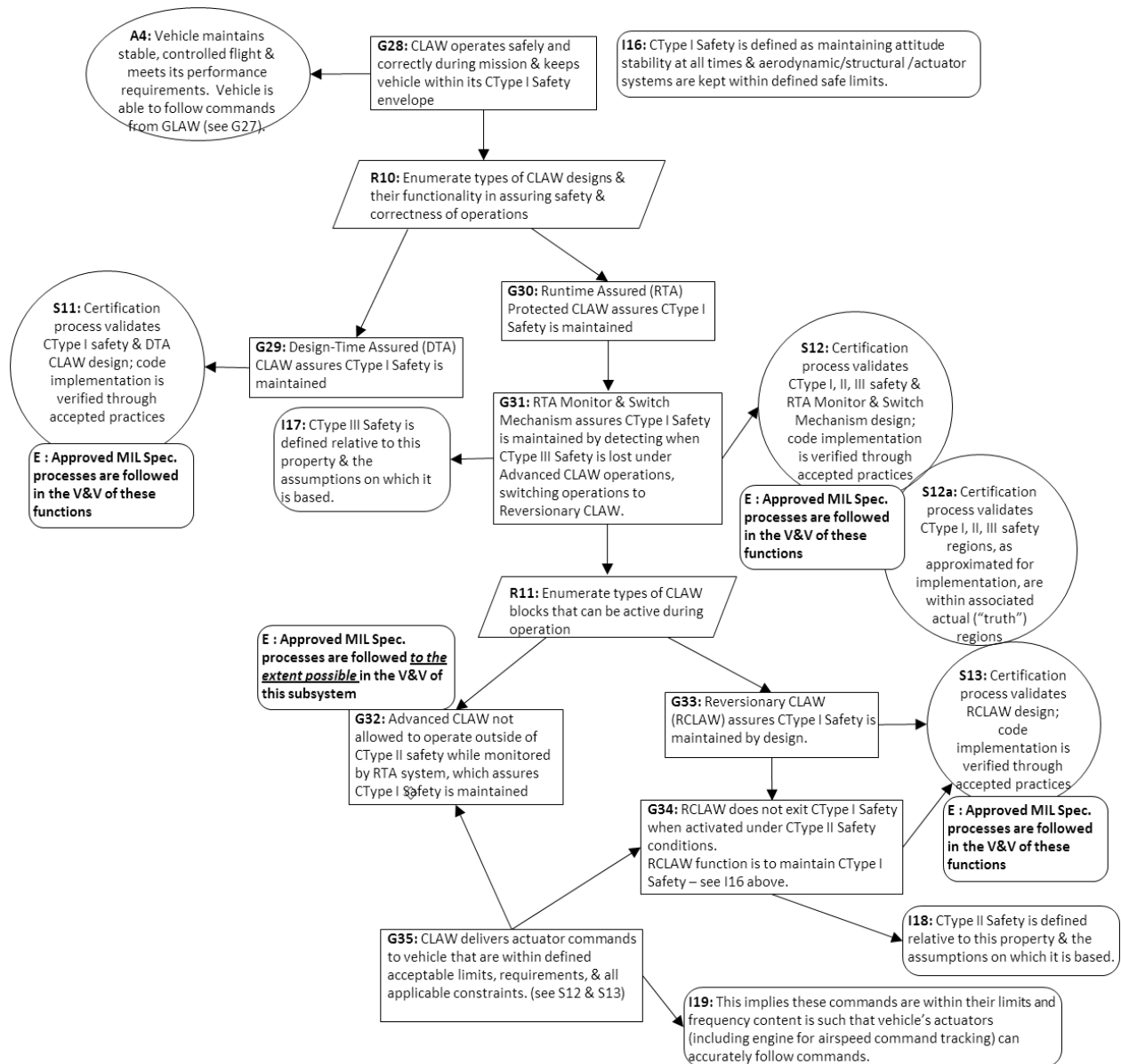


Figure 88. Safety Argument for CLAW Feedback Level

## 11 Alternative Frameworks

We have provided particular recommended approaches to the development of RTA protected frameworks in this report. However, some alternative approaches warrant consideration for follow on efforts and these are discussed here.

### 11.1 Integrated RTA Protected Feedback Loops

As was discussed in Section 4.5, we have recommended a modular framework here in which each feedback level can have a DTA or RTA protected system and that the input/output structure of the advanced, reversionary and DAT systems must be the same so that any one of them can take over operation or control at any time. This leads to a *plug and play* type architecture where the input/output mapping is made universal. This modular framework certainly has its benefits. However, an alternative framework is one in which the upstream and downstream advanced systems are tightly coupled, as well as the reversionary upstream/downstream systems. This can have certain performance advantages in that each upstream advanced system is generating commands only for its downstream advanced components. The disadvantage here is the lack of modularity. This framework would most likely only be undertaken if there was only one single design team or organization that was developing and constructing all feedback levels of the complete system. The other disadvantage is that if one feedback level reverts, they all must revert since each reversionary system can only accept commands from the reversionary system at the next level up, and can only deliver commands to the reversionary system at the next feedback level down. It is recommended that further study be performed if there is interest in a fully integrated approach.

### 11.2 Allowing Post Reversion Restart of Advanced Systems

We have recommended here that if the advanced system is deemed to be working incorrectly because of ensuing unsafe conditions, then it should be shut down permanently because the advanced system can no longer be trusted. In other applications, however, this may not necessarily be the best approach. In a related NASA project [Schierman 2014(a)] we investigated the application of RTA to advanced aircraft propulsion control. One key finding is that such engine systems often operate at or near their limits and critical monitored parameters will often exceed their designated limits during transient operations as the system transitions from one operating point to another (usually due to changes in throttle). However, these exceedances are temporary and are not necessarily an indication that some error exists in the advanced controller. In fact, the engine control community has long used *power management systems* which are analogous to an engine governor. In general, power management systems will adjust the fuel flow rate (up or down) to maintain steady, stable combustion when certain critical parameters exceed their upper or lower bounds (e.g., surge margins). These power management systems are clearly related to RTA systems, although they are used even for trusted, certified control systems as well.

For engine systems, therefore, completely shutting down the advanced controller at the first instance of a critical parameter violating its bound may be an overly conservative switching protocol. Instead, some additional logic should be developed to determine if the bound violation is temporary due to transient operations or is indeed due to errors in the advanced controller. One method that we explored was to have the RTA system switch back to the advanced

controller after all measured parameters were back within their allowable operating ranges. This approach worked well for the engine system we investigated and eliminated RTA *false alarm* occurrences. In summary, switching protocols can be different for different applications.

A key point here is that the RTA monitor was still in use after the mode reversion was activated. That is, the RTA monitor is not necessarily a *one-time trigger* that is no longer in use once the system is switched to reversionary control – even if the protocol is to never switch back to the advanced system. For example, the global RTA Manager may still require information from the RTA monitors at the various levels, even if some of those levels have already activated mode reversion.

### **11.3 Diagnostic and Prognostic Checking for Post Mission Analysis**

If further analysis of the advanced system can be performed during runtime, then additional useful information may be gained. Software or design errors in the advanced elements may exist that do not cause immediate or observable adverse effects during design time testing or initial runtime operations. However, over longer periods of the operational life of the platform, such errors may begin to produce degraded performance, reductions in safety margins, or may lead to hardware maintenance issues, etc. Therefore, if possible with the onboard computing resources available, it is proposed that diagnostic or prognostic checking/analysis be performed. Such analysis may take on a variety of forms and clearly will be application specific. Formal methods approaches may provide key analysis attributes for runtime checking. A given set of software artifacts may be checked against a given set of formalized properties during system operation. Note also that statistical isolation methods used in FDI approaches may apply here to determine where the software error resides. The difference, of course, is that FDI approaches attempt to determine hardware faults, whereas here, we are attempting to determine software faults. Some diagnostic properties may be defined such that if violated at runtime but safety or performance is not compromised, then the event is simply recorded and tracked for further post mission analysis. Certain critical parameter values may vary over time and indicate some type of degradation, for example. Or, a certain event may occur at a more frequent rate over the course of several missions, signifying a certain class of error exists in the system. Prognostic checking may also be performed by monitoring the advanced system's performance over time and determining if and when identified *minor* software errors may eventually lead to critical operational problems or system instability.

## 12 Conclusions

This report has presented candidate architectures for the RTA system as it is integrated into the overall feedback system for aerospace platforms. This development explored interacting RTA systems in a nested feedback framework and address issues and limitations noted from our prior work.

We developed a challenge problem to address interest in complex autonomy of future Air Force systems. This problem involved a multi-vehicle fleet of UASs performing some type of complex mission, working together in a distributed, cooperative command and control protocol. One of the key contributions of this program centered on how the nested feedback loops interact and influence each other and the implications of these interactions on their RTA system designs. This program developed a unique, modular framework in which each feedback level could either be an industry standard design time assured system, or could be a RTA protected system that could be in either advanced or reversionary operational mode. The implications of switching from advanced to reversionary mode at one level and the effects that can have on other levels was explored. It was determined that the RTA monitoring process should involve more than just checking for system safety at the current feedback level, as degraded system performance at one level can have safety implications at other levels due to the inherent interdependence of each feedback level. Because of this, it was determined that an overall RTA monitor manager needs to continually pass operating mode information to each of the feedback levels so that each level is always aware of whether other feedback levels are operating in advanced or reversionary mode. Reversionary systems may have reduced performance capabilities, and this information needs to be communicated with the other feedback levels to ensure overall correct and safe operation.

Another main contribution from this effort involved developing a reversionary system composed of any number of transition controllers or systems, and any number of baseline controllers or systems, whatever is required to fully cover the operating region of the advanced system. We explored design methods for inner-loop transition controllers that could guarantee progress of the state from the switching condition boundary to the operating region of a chosen baseline controller.

Our past efforts focused on developing RTA systems in isolation, applied first to an inner-loop control system and then second to an outer-loop autoland guidance system. Another main contribution here was to expand our RTA applications to higher level feedback loops, namely the flight management system and the fleet mission planning system. Our preliminary designs of these systems were used to explore RTA protection at these higher loops and how they interact with the more familiar lower level guidance and control loops.

Another key contribution was the refinement of what we termed type safety definitions. These formal definitions of safety levels addressed some of the key research questions that were left unresolved from our previous RTA programs regarding when to switch to the reversionary system and always be assured of safe operation.

Because of the complexity of advanced systems, we investigated an analysis approach known as compositional reasoning, which involves the construction of assume-guarantee (A-G) contracts for sub-elements or subsystems within the overall feedback system of the aircraft platform.

Another key contribution of this project was the recognition that RTA systems must act as A-G contract monitors, ensuring that these contracts are always upheld. If there is a breach of contract for whatever reason, then this breach is mitigated by employing the appropriate reversionary controller or reversionary actions. The A-G contract checks that must be performed by the RTA monitor involve more than just safety checks at the current feedback level. They must also include required performance checks, checks on input and output validity, and hardware status checks including sensor and information integrity checks. These additional checks are required due to the complex inter-connectivity of the integrated, nested feedback architecture we addressed in this project's challenge problem.

One of the most important considerations for future autonomous aerospace systems is their safe integration into airspaces that include manned or piloted aircraft, as well as operations over congested, populated regions. Trusted automated ground and airborne collision avoidance systems will be essential to enable this integration. Another contribution of this effort was to investigate how to best design a framework that integrates interacting certified collision avoidance systems with the nested, hierarchical RTA protected framework. Our recommended approach is to have a completely separate and dedicated collision avoidance feedback framework that takes control of the vehicle at all feedback levels when a collision threat is detected.

Another set of contributions accomplished in this effort addressed the design time certification of the RTA protected framework. We developed prototype fault tree analysis (FTA) and failure mode effects and criticality analysis (FMECA) models that directly addressed the fault mitigation functions of the RTA systems. We also used the results of the compositional reasoning and A-G contract construction to form the basis for a preliminary safety case argument using goal structuring notation (GSN) diagrams. Here we developed an argument pattern that could be applied at each feedback level. It is hoped that constructing safety case arguments will aid in developing specific artifacts and evidence that can be used in the required certification processes for eventual fielding of RTA protection on complex systems.

Certification of new and novel ideas is fraught with challenges in traditional, conservative safety critical domains. We applied interacting RTA protected systems to a highly complex, hierarchical feedback framework of interacting platforms. The complexity of the system itself resulted in complexity of the overall set of interacting RTA systems, with many technical issues that required careful consideration. In summary, complex systems will require complex RTA protection. For RTA protected systems to be implemented in future systems, we recommend that incremental progress should first be made with these new ideas by introducing complexity in small steps. We suggest:

1. Introduce a single or isolated advanced system with RTA protection as a *final emergency option* or as a *last layer of protection* only when all else fails, and only first in remote area operations.
2. When no problems are seen in this initial introduction (or all noted problems have been addressed and fixed), introduce the isolated advanced system with RTA protection as the *primary option*, but again only in remote area operations.
3. When this application of RTA becomes trusted and accepted, introduce the RTA concept in a limited operational environment where there are fewer safety challenges.

4. Begin to introduce more complexity in the RTA framework with more complex applications. With each new application, graduate to more challenging environments and use cases when the current application shows no signs of problems and becomes more trusted.

In each of these steps of increasing complexity, the list of assumptions, constraints and the information base of *known* hazards will grow, thus leading to more robust solutions with more plausible safety cases. The technical and certification hurdles will then begin to come down as more trust in RTA protection grows.



## 13 Recommendations

Design, development and implementation of RTA systems are not solved problems. A number of key areas require further maturation. First, developing feasible switching condition boundary construction techniques will be required. Although we made great progress in formally defining the switching condition objectives, we have yet to solve the curse of dimensionality associated with the highly complex, multi-dimensional nature of the hypersurface boundaries that define the reversionary switching decision. Currently, this requires labor intensive, costly simulation exercises and extensive analyses. However, there seem to be promising mathematical approaches and software tools for reducing this burden and it is recommended that these approaches be further pursued. Some of these areas include continuation methods, support vector machines, signed distance functions, and reach and backreach approximations.

Our approach has been to develop techniques for constructing the switching condition boundaries offline and then query the measured set of states and critical parameters online to determine if switching to the reversionary system is warranted. However, online prediction schemes and state reachability analysis may hold promise in more accurately determining when the reversion switch should be executed. We have explored simplified linear prediction schemes in our past Phase II program and have analyzed some software tools for state reachability in this program. Appendix F presents a summary of our findings. However, several researchers in the field are actively pursuing the state reachability problem to aid in alleviating cumbersome simulation approaches and it is recommended that work in this area continue in collaboration with these researchers.

Related to the curse of dimensionality problem is the development of methods for streamlining construction of reversionary systems. If the design and development of the reversionary systems are required to be so complicated that they negate the advantages of using the advanced system, then this will severely limit the benefits of the RTA protection and risk mitigation approaches addressed in this program. We directly addressed this problem at the inner-loop level and provided some preliminary approaches, but recommend further attention be paid to this area of research and development.

The nested RTA protected feedback loops have a high degree of interaction and the implications of switching to reversionary modes at one or more levels should be explored further. If some of the feedback levels are allowed to continue to run in advanced mode, while other levels are running in reversionary mode, then this framework of *graceful degradation* can have the potential for many unforeseen emergent behaviors, especially for heterogeneous fleets operating in any number of mixed-mode combinations. This issue can lead to unexpected behaviors in the fleet, especially if a number of sub-fleets separate, perform sub-missions and then rendezvous and regroup. In this case, different vehicles in the fleet experienced different mission events and different environments, which could lead to mode reversion at certain feedback levels for some portion of the fleet whereas the rest of the fleet remains unaffected. These scenarios need to be carefully explored before such systems can be certified. It is recommended to explore this topic further and to construct guidelines for how such disparate platforms can seamlessly work together in either advanced or reversionary modes at different feedback levels.

The complexity of the systems that we explored for RTA protection resulted in complex RTA designs themselves. If such systems are to eventually be certified for operational deployment, then the topic of certifying complex RTA protected systems must continue to be addressed. We recommend meeting with more FAA DERs and counterpart DoD certification authorities to alert these governing bodies to the ideas of RTA protection and to seek their feedback and guidance on how to construct RTA systems that can be design-time certified.

## 14 References

- Aguiar, A., Kaminer, I., Ghabcheloo, R., Pascoal, A., Xargay, E., Hovakimyan, N., Cao, C., Dobrokhodov V., “Time-Coordinated Path Following of Multiple UAVs Over Time-Varying Networks Using L1 Adaptation,” Proc. AIAA Guidance, Navigation and Control Conference, AIAA 2008-7131, Honolulu, HI, Aug., 2008.
- Aiello, A., Berryman, J., Grohs, J., Schierman, J., “Run-Time Assurance for Advanced Flight-Critical Control Systems,” Proc. AIAA Guidance, Navigation, and Control Conference, AIAA 2010-8041, Toronto, Ontario Canada, Aug., 2010.
- Arehart, A., Wolovich, W., “Bumpless Switching Controllers,” Proc. IEEE Conference on Decision and Control, Dec., 1996, pp. 1654–1655.
- ARP4754A, anon., Aircraft and System Development Processes, Washington, D.C., SAE Aerospace International, 2010.
- ARP4761, anon., Safety Assessment Process Guidelines and Methods, Washington, D.C., SAE Aerospace International, 1996.
- Bak, S., *et. al.* “Using Run-Time Checking to Provide Safety and Progress for Distributed Cyber-Physical Systems,” Proc. IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2013.
- Bak, S., Greer, A., Mitra, S., “Hybrid Cyberphysical System Verification with Simplex Using Discrete Abstractions,” Proc. 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010.
- Bak, S., Johnson, T., Caccamo, M., Sha, L., “Real-Time Reachability for Verified Simplex Design,” 35th IEEE Real-Time Systems Symposium, 2014.
- Bak, S., Manamcheri, K., Mitra, S., Caccamo, M., “Sandboxing Controllers for Cyber-Physical Systems,” Proc. IEEE/ACM International Conference on Cyber-Physical Systems (ICCPs), April, 2011, pp. 3-12.
- Beard, R., McLain, T., *Small Unmanned Aircraft - Theory and Practice*, Princeton University Press, Princeton and Oxford, March 2011.
- Benvenuti, L., Bresolin, D., Casagrande, A., Collins, P., Ferrari, A., Mazzi, E., Sangiovanni-Vincentelli, A., Villa, T., “Reachability Computation for Hybrid Systems with Ariadne,” Proc. of the 17th IFAC World Congress, pp. 8960–8965, Seoul, Korea, 2008.
- Boccaro, N. *Modeling Complex Systems*, Springer-Verlag, New York, 2004.
- Boids, <http://www.vergenet.net/~conrad/boids/pseudocode.html>, 2002.
- Boskovic, J., Knoebel, N., Moshtagh, N., Larson, G., Amin, J. “Collaborative Mission Planning & Autonomous Control Technology (CoMPACT) System Employing Swarms of UAVs,” Proc. AIAA Guidance, Navigation, and Control Conference, AIAA 2009-5653, Chicago, Ill., Aug., 2009.
- Buffington, Crum, Krogh, Plaisted, Prasanth, Bose, Johnson, “Validation and Verification of Intelligent and Adaptive Control Systems,” Proc. 2nd AIAA Unmanned Unlimited Conf. and Workshop & Exhibition, San Diego, CA, Sept., 2003.
- Buhr, B., “Use Case Maps as Architectural Entities for Complex Systems,” Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada, Aug., 1998.

- Chazelle, B., "Natural Algorithms and Influence Systems," *Communications of the ACM* vol. 55, no. 12 (Dec. 2012), pp. 101-110.
- Chen, X., Abraham, E., Sankaranarayanan, S., "Taylor Model Flowpipe Construction for Non-linear Hybrid Systems," *Proc. of the 2012 IEEE 33rd Real-Time Systems Symposium (RTSS)*, IEEE Computer Society Press, 2012.
- Clark, M., Koutsoukos, X., Kumar, R., Lee, I., Pappas, G., Pike, L., Porter, J., Sokolsky, O., "A Study on Run Time Assurance for Complex Cyber Physical Systems," AFRL Final Report, April, 2013.
- Cofer, D., "Complexity-Reducing Design Patterns for Cyber-Physical Systems," AFRL Final Report: AFRL-RZ-WP-TR-2011-2098, Sept., 2011.
- Cofer, D., Gacek, A., Miller, S., Whalen, M., LaValley, B., Sha, L., "Compositional Verification of Architectural Models," In *NFM 2012, LNCS 7226*, by A., Person, S. Goodloe, 126-140. Berlin & Heidelberg: Springer-Verlag, 2012.
- Cooper, J., Richards, N., Schierman, J., Neal, D., Lichter, M., Schlapkohl, T., Su, S., "A Sense and Avoid System for Unmanned Aircraft in Formation Flight," *Proc. AIAA SciTech Forum, AIAA2014-0967*, National Harbor, MD, Jan., 2014.
- Dalmao, F., Mordecki, E., "Cucker-Smale Flocking Under Hierarchical Leadership and Random Interactions," *SIAM Journal of Applied Mathematics*, vol. 71, no. 4, (2011), pp. 1307-1316.
- Dang, T., Maler, O., "Reachability Analysis via Face Lifting," *Proc. First International Workshop on Hybrid Systems: Computation and Control (HSCC '98)*, vol. 1386 of *Lecture Notes in Computer Science*, pp. 96-109, Springer, 1998.
- Despotou, G., GSN Reference Card, v. 1.2, University of York, York, 2010.
- DeVore, M., Bateman, A., "Polynomial Chaos Theory for Performance Evaluation of ATR Systems," *Proc. SPIE*, 7696, 76960R. 2010.
- DeVore, M., Schierman, J., Lichter, M., "Methodology and Tools For Certifying Autonomous, Cyber-Physical Systems-Of-Systems," AFRL Final Report, AFRL-RQ-WP-TR-2014-0068, March, 2014.
- DO-178C, anon., *Software Considerations in Airborne Systems and Equipment Certification*, Washington, D.C., RTCA, 2011.
- DoD, anon., *Unmanned Aircraft System Airspace Integration Plan*, Department of Defense, March, 2011.
- DoD, anon., *Memorandum: Autonomy Test and Evaluation, Verification and Validation Technology Investment Strategy 2015-2018*, Office of the Secretary of Defense, 2015.
- Dutta, K. "How Birds Fly Together: The Dynamics of Flocking," *Resonance*, Dec. 2010, pp. 1097-1110.
- Frazzoli, E., Dahleh, M., Feron, E., "Real-Time Motion Planning for Agile Autonomous Vehicles," *Journal of Guidance, Control and Dynamics* 25 (Jan. - Feb. 2002), pp. 116-129.
- Gandhi, N., Cooper, J., Ward, D., "Scalable Approaches to Integrated Flight and Planform Control of Morphing Air Vehicles," Final Report, Contract FA8650-06-C-3608, AFRL SBIR Phase II, September, 2008.
- Graebe, S., Ahl'n, A., "Dynamic Transfer Among Alternative Controllers and its Relation to Antiwindup Controller Design.," *IEEE Transactions on Control Systems Technology*, Vol. 4, no. 1 (Jan. 1996), pp. 92-99.

- Gregory, I., Cao, C., Xargay, E., Hovakimyan, N., Zou, X., "L1 Adaptive Control Design for NASA AirSTAR Flight Test Vehicle," Proc. AIAA Guidance, Navigation, and Control Conference, AIAA 2010-8015, Chicago, IL, Aug., 2009.
- GSN, anon., GSN Community Standard Version 1, York: Origin Consulting Limited, Nov., 2011.
- Hollingsworth, M., Cotton, G., Cofer, D., Schierman, J., Wagner, L., Aiello, A., Grohs J., "Certification Techniques for Advanced Flight Critical Systems Challenge Problem Demonstration (CerTA FCS CPD)," Final Report, Lockheed Martin Corporation, AFRL-RB-WP-TR-2010-3039, May, 2010.
- Humphrey, L., Model Checking for Verification in UAV Cooperative Control Applications, Vol. 444, chap. 4 in *Recent Advances in Research on Unmanned Aerial Vehicles*, by F. Fahroo, Lecture Notes in Control and Information Sciences, 2013, pp. 69-117.
- Humphrey, L., "Model Checking UAV Mission Plans." Proc. AIAA Modeling and Simulation Technologies Conference. AIAA 2012-4723, Minneapolis, MN., Aug., 2012.
- Joel, J., Karaman, S., Frazzoli, E., "Anytime Computation of Time-Optimal Off-Road Vehicle Maneuvers using the RRT\*," IEEE Conf. on Decision and Control, 2011.
- Johnson, C., "What are Emergent Properties and How Do They Affect the Engineering of Complex Systems," Dept. of Computing Science, University of Glasgow, Glasgow Scotland, U.K., 2008.
- Johnson, T., "Stability Analysis of Simplex Architecture Controlled Inverted Pendulum." Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign., <http://www.academia.edu/276649/>, 2008.
- Johnson, T., Mitra, S. Safe Flocking in Spite of Actuator Faults. Vol. vol. 6366, in 12th International Symposium on Stabilization, Safety, and Security of Distributed Systems, pp. 588-602. Berlin / Heidelberg: Springer, Sept., 2010.
- Joint Chiefs of Staff, anon., Command and Control of Joint Air Operations, Washington, DC, Department of Defense, February, 2014.
- JUAS, anon., "COE CONOPS Joint Concept of Operations for Unmanned Aircraft Systems," Chapter 2, Version 1.5, 2009.
- Kaminer, I., et al., "Path Following for Unmanned Aerial Vehicles Using L1 Adaptive Augmentation of Commercial Autopilots," *AIAA Journal of Guidance, Control, and Dynamics*, Vol. 33, no. 2 (March-April 2010).
- Karaman, S. and Frazzoli, E., "Sampling-based Optimal Motion Planning for Non-holonomic Dynamical Systems," IEEE Conference on Robotics and Automation (ICRA), 2013.
- Karaman, S. Walter, M., Perez, A., Frazzoli, E., Teller, S., "Anytime Motion Planning using the RRT\*," 2011 IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China, May, 2011.
- Karaman, S., Frazzoli, E., "Optimal Kinodynamic Motion Planning using Incremental Sampling-based Methods," Proc. 49th IEEE Conference on Decision and Control. Atlanta, Dec., 2010.
- Karaman, S., Frazzoli, E., "Vehicle Routing with Linear Temporal Logic Specifications: Applications to Multi-UAV Mission Planning," Proc. AIAA Guidance, Navigation and Control Conference. AIAA 2008-6838, Honolulu, HI, Aug., 2008.

- Karaman, S., Frazzoli, E., "Incremental Sampling-based Optimal Motion Planning," Robotics: Science and Systems, 2011 (a).
- Karaman, S., Frazzoli, E., "Sampling-based Optimal Motion Planning for Non-holonomic Dynamical Systems," IEEE Conference on Robotics and Automation (ICRA), 2013 (a).
- Kavraki, L., Svestka, P., Latombe, J., Overmars, M., "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces." IEEE Transactions on Robotics and Automation 12 (1996): 566-580.
- Ke, Y., Tsourdos, A., White, B., Silson, P., "Dynamic Mission Planning of Multiple Unmanned Autonomous Platforms for Sensing and Autonomous Urban Reconnaissance," Proc. AIAA Guidance, Navigation, and Control Conference, AIAA 2009-6216, Chicago, Ill., Aug., 2009.
- Kingston, D., Rasmussen, S., Mears, M., "Base Defense Using a Task Assignment Framework," Proc. AIAA Guidance, Navigation, and Control Conference, AIAA 2009-6209, Chicago, Ill., Aug., 2009.
- Kuwata, Y., Teo, J., Fiore, G., Karaman, S., Frazzoli, E., How, J., "Real-Time Motion Planning with Applications to Autonomous Urban Driving," *IEEE Transactions on Control Systems Technology*, Jan. 2009.
- Lamont, G., UAV Swarm Mission Planning Development Using Evolutionary Algorithms and Parallel Simulation - Part II, Report SCP-195, Wright-Patterson AFB, OH: Department of Electrical and Computer Engineering, Graduate School of Engineering and Management, Air Force Institute of Technology, 2008.
- LaValle, S., Kuffner, J. "Randomized Kinodynamic Planning," *International Journal of Robotics Research*, 20, no. 5 (May 2001), pp. 378-400.
- Liu, X., et al., "ORTEGA: An Efficient and Flexible Online Fault Tolerance Architecture for Real-Time Control Systems," *IEEE Transactions on Industrial Informatics*, 4(4), pp. 213-224 2008.
- Luders, B., Karaman, S., Frazzoli, E., How, J., "Bounds on Tracking Error using Closed-Loop Rapidly-Exploring Random Trees," Proc. American Controls Conference. Baltimore, MD., 2010.
- Majumdar, A., Tedrake, R., "Robust Online Motion Planning with Reachable Sets," S.M. Thesis, MIT, 2013.
- Manson, S., "Validation and Verification of Multi-Agent Models for Ecosystem Management," Theory and Practice of Multi-Agent Approaches, 2003, pp. 63-74.
- Marshall, C., Mears, M., Kingston, D., Rasmussen, S., "2010 ICE-T Cooperative Control Flight Testing." Proc. Infotech@Aerospace, AIAA 2011-1486, St. Louis, MO., March, 2011.
- Mataric, M., "Designing Emergent Behaviors: From Local Interactions to Collective Intelligence," Proc. Simulation of Adaptive Behavior, MIT Artificial Intelligence Laboratory: J-A. Meyer, H. Roitblat and S. Wilson, eds., MIT Press, 1993, pp. 432-441.
- Mogul, J. "Emergent (Mis) Behavior vs. Complex Software Systems," Proc. of the EuroSys. HP Labs, Palo Alto, 2006, pp. 293-304.
- Oehlerking, J., Theel, O., "Decompositional Construction of Lyapunov Functions for Hybrid Systems," Proc. 12th International Conference on Hybrid Systems: Computation and Control (HSCC 2009), pp. 276-290.

- OSD, anon., Unmanned Aircraft Systems Roadmap 2005-2030, Office of the Secretary of Defense, Aug., 2005.
- Parunak, H., VanderBok, R. "Managing Emergent Behavior in Distributed Control Systems," presented at ISA-Tech, Anaheim, CA, 1997.
- Platzer, A., Clarke, E., "Computing Differential Invariants of Hybrid Systems as Fixedpoints," Proc. 20th International Conference on Computer Aided Verification (CAV 2008), pp. 176-189.
- Pnueli, A., Siegel, M., Singerman, E., "Translation Validation," Proc. 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '98), vol. 1384 of Lecture Notes in Computer Science, Springer-Verlag, 1998.
- Prajna, S., Jadbabaie, A., "Safety Verification of Hybrid Systems Using Barrier Certificates," Proc. 7th International Workshop on Hybrid Systems: Computation and Control (HSCC), vol. 2993 of Lecture Notes in Computer Science, pp. 477-492, Springer-Verlag, 2004.
- Prajna S., Jadbabaie, A., Pappas, G., "A framework for Worst-Case and Stochastic Safety Verification using Barrier Certificates," *IEEE Transactions on Automatic Control*, 52(8), pp. 1415-1429, August 2007.
- Protti, M., Barzan, R., "UAV Autonomy – Which level is desirable? – Which level is acceptable? Alenia Aeronautica Viewpoint," Final Report: Alenia Aeronautica, RTO-MP-AVT-146, Nov., Torino, Italy, 2007.
- Richards, N., Bird, R., "UAV 2D and 3D Path Planning for Sensor-On-Target Maneuvers and Obstacle Avoidance," Barron Associates Technical Report 294; prepared for Northrop Grumman Software Enabled Control, 2004.
- Rivera, J., Danylyszyn, A., Weinstock, C., Sha, L., Gagliardi, M., "An Architectural Description of the Simplex Architecture," Carnegie Mellon University Technical Report CMU/SEI-96-TR-006 ESC-TR-96-006, March, 1996.
- Rubio, J., Vagners, J., Rysdyk, R., "Adaptive Path Planning for Autonomous UAV Oceanic Search Missions," Proc. AIAA 1st Intelligent Systems Technical Conference. AIAA 2004-6228, Chicago, Ill., Sept., 2004.
- Rudd, L., "Switch Control Architecture for Advanced Control System Certification," Proc. AIAA Guidance, Navigation, and Control Conference, AIAA-2009-5674, Chicago, IL, Aug., 2009.
- Sankaranarayanan, S., Sipma, H., Manna, Z., "Constructing Invariants for Hybrid Systems," *Formal Methods in System Design*, 32(1), pp. 25-55, 2008.
- Schierman, J., DeVore, D., Lichter, M., Mitra, S., Topcu, U., "Combined Approaches for Verification and Validation of Run Time Protected Systems," AFRL Final Report, AFRL-RQ-WP-TR-2014-0211, Sept., 2014.
- Schierman, J., Horneman, K., O'Brien, C., Neal, D., Su, S., Ehlmann, K., Adams, R., Healy, P., Lin, J., "Rapid Automated Flight Planning for High Speed Systems," Barron Associates, Inc., Final Technical Report, AFRL-RB-WP-TR-2012-0164, 2012.
- Schierman, J., Schlapkohl, T., "Run Time Assurance Methods Applied to Advanced Propulsion Algorithms," NASA Glenn Research Center Contract No. NNC12CA12C, Final Report SBIR PHASE III, April, 2014(a).

- Schierman, J.D., Ward, D.G., Dutoi, B.C., et al., "Run-Time Verification and Validation for Safety-Critical Flight Control Systems," Proc. AIAA Guidance, Navigation, and Control Conference, AIAA-2008-6338, Honolulu, HI, Aug., 2008.
- Schouwenaars, T., DeMoor, B., Feron, E., How, J., "Mixed Integer Programming for Multi-Vehicle Path Planning," Proc. of the European Control Conference, Sept., 2001.
- Seto, D., Krogh, B., Sha, L., Chutinan, A., "The Simplex Architecture for Safe Online Control System Upgrades," Proc. American Control Conference. Philadelphia, PA, June, 1998, pp. 3504-3508.
- Seto, D., Sha, L., "An Engineering Method for Safety Region Development, Paper 137," Software Engineering Institute, 1999, <http://repository.cmu.edu/sei/137>.
- Sha, L., "Using Simplicity to Control Complexity," *IEEE Software* 18(4), pp. 20-28, July/August 2001.
- Sha, L., Goodenough, J., Pollak, B., "Simplex Architecture: Meeting the Challenges of Using COTS in High-Reliability Systems," CROSSTALK The Journal of Defense Software Engineering, April 1998.
- Sha, L., Rajkumar, R., Gagliardi, M. "A Software Architecture for Dependable and Evolvable Industrial Computing Systems," Carnegie Mellon University Technical Report CMU/SEI-95-TR-005 ESC-TR-95-005, July, 1995.
- Speltzer, M., Narang-Siddarth, A., "Continuation Analysis of Nonlinear Systems with Equality Constraints on States, Parameters and Eigenvalues," Proc. AIAA SciTech Forum, Kissimmee, FL, Jan., 2015.
- Srivastava, A., Integrated Vehicle Health Management, Technical Plan, Version 2.03. NASA, 2009.
- Stevens, B., Lewis, F., *Aircraft Control and Simulation*, John Wiley & Sons, Inc., New York, 1992.
- Stenger, A., Fernando, B., Heni, M., "Autonomous Mission Planning for UAVs: A Cognitive Approach," Deutscher Luftund Raumfahrtkongress, Document ID: 281398, 2012.
- Storm, W., Whalen, M., Cofer, D., and Krogh, B., "Certification Techniques for Advanced Flight Critical Systems," Lockheed Martin Aeronautics Company Report No. FZM-9465, June, 2008.
- Sucan, I., Moll, M., Kavraki, L., "The Open Motion Planning Library," IEEE Robotics & Automation Magazine, Dec. 2012: 72-82.
- Sullivan, D., et al., "Intelligent Mission Management for Uninhabited Aerial Vehicles," Proc. SPIE Conference and Exhibit, 2004.
- TEVV, anon., "DoD Autonomy COI Test and Evaluation, Verification and Validation (TEVV) Investment Strategy 2015-2018," Assistant Secretary of Defense, Research and Engineering (ASD/R&E), June 2015. Available online at: [http://www.defenseinnovationmarketplace.mil/resources/OSD\\_ATEVV\\_STRAT\\_DIST\\_A\\_SIGNED.pdf](http://www.defenseinnovationmarketplace.mil/resources/OSD_ATEVV_STRAT_DIST_A_SIGNED.pdf)
- Tobenkin, M., Manchester, I., Tedrake, R., "Invariant Funnels around Trajectories using Sum-of-Squares Programming," Proc. 18th IFAC World Congress, Milano, Italy, 2011.
- Torens, C., Adolf, F., "Automated Verification and Validation of an Onboard Mission Planning and Execution System for UAVs," Proc. AIAA Infotech@Aerospace (I@A) Conference. AIAA 2013-4564, Boston, MA., Aug., 2013.



- Vandenberghe, L., Boyd, S., Wu, S., “Determinant Maximization with Linear Matrix Inequality Constraints,” *SIAM Journal on Matrix Analysis and Applications*, 19(2) pp. 499-533, 1998.
- Wadley, J., “Development of an Automatic Aircraft Collision Avoidance System for Fighter Aircraft,” Proc. AIAA Infotech@Aerospace (I@A) Conference, AIAA 2013-4727, Boston, MA, Aug., 2013.
- Waitz, I. “Lecture#2, The Breguet Range Equation,” Open Courseware M.I.T. Lecture Notes (Fall). 2003. ocw.mit.edu.
- Ward, D., Schierman, J., Dutoi, B., Aiello, M., Berryman, J., Grohs, J., DeVore, M., Storm, W., Wadley, J., Tallant, G., “Run-Time Validation and Verification (V&V) for Safety-Critical Flight Control Systems,” AFRL Final Report, AFRL-RB-WP-TR-2009-3071, March, 2009.
- wikiGPS, anon., GPS Exchange Format, 2014,  
[http://en.wikipedia.org/wiki/GPS\\_Exchange\\_Format#Data\\_types](http://en.wikipedia.org/wiki/GPS_Exchange_Format#Data_types).
- Yao, J., Liu, X., Zhu, G., Sha, L., “NetSimplex: Controller Fault Tolerance Architecture in Networked Control Systems,” *IEEE Transactions on Industrial Informatics* 9(1), pp. 346-356, 2013.
- Yong, E., “How the Science of Swarms Can Help Us Fight Cancer and Predict the Future,” *Wired Magazine*, March 2013.

## LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS

|        |                                                                           |
|--------|---------------------------------------------------------------------------|
| AAP    | asset allocation plan                                                     |
| ACLAW  | advanced control law                                                      |
| ADS    | air data system                                                           |
| ADS-B  | automatic dependent surveillance-broadcast                                |
| AFRL   | Air Force Research Laboratory                                             |
| AFMS   | advanced flight management system                                         |
| A-GCAS | automatic ground collision avoidance system                               |
| AGLAW  | advanced guidance law                                                     |
| AGREE  | assume guarantee reasoning environment                                    |
| AIS    | automatic identification system                                           |
| AMPS   | advanced mission planning system                                          |
| ATO    | air tasking order                                                         |
| AWP    | advanced waypoint                                                         |
| AVO    | air vehicle operator                                                      |
| BAI    | Barron Associates, Inc.                                                   |
| BLOS   | beyond line-of-sight                                                      |
| C3     | command/control/communications                                            |
| CA     | collision avoidance                                                       |
| CBRNE  | communications relay & chemical biological radiological nuclear explosive |
| CCA    | common cause analysis                                                     |
| CCAS   | certified collision avoidance system                                      |
| CLAW   | control law                                                               |
| CDRL   | contract data requirements list                                           |
| DAL    | design assurance level                                                    |
| DER    | designated engineering representative                                     |
| DFTR   | draft final technical report                                              |
| DoD    | Department of Defense                                                     |
| DOF    | degrees of freedom                                                        |
| DTA    | design time assurance                                                     |
| EO/IR  | electro-optical/infrared                                                  |
| FAA    | Federal Aviation Administration                                           |
| FHA    | functional hazard assessment                                              |
| FMECA  | failure mode effects and criticality analysis                             |
| FDIR   | fault detection, isolation and recovery                                   |
| FM     | formal methods                                                            |
| FMS    | flight management system                                                  |
| FOV    | field of view                                                             |
| FTA    | fault tree analysis                                                       |
| FTR    | final technical report                                                    |
| GA     | general aviation                                                          |
| GLAW   | guidance law                                                              |
| GPS    | global positioning system                                                 |
| GPX    | GPS exchange format                                                       |
| HIL    | hardware-in-the-loop                                                      |
| ICD    | interface control document                                                |

## LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS (concluded)

|        |                                                         |
|--------|---------------------------------------------------------|
| IDEA   | integration design, evaluation, and assurance           |
| IMP    | initial mission plan                                    |
| IMU    | inertial measurement unit                               |
| INS    | inertial navigation system                              |
| ISR    | intelligence, surveillance, reconnaissance              |
| IVHM   | integrated vehicle health management system             |
| KPP    | key performance parameter                               |
| LRF/D  | laser range finder/designator                           |
| LTL    | linear temporal logic                                   |
| MAS    | morphing air structures                                 |
| MM     | mission management                                      |
| MML    | mission modeling language                               |
| MoM    | measure of merit                                        |
| MPS    | mission planning system                                 |
| MS     | mission segment                                         |
| MSR    | monthly status report                                   |
| MUSAA  | multi-vehicle unmanned aircraft systems sense and avoid |
| NAS    | national airspace system                                |
| NAV    | navigation system                                       |
| OR     | operational region                                      |
| PECVD  | plasma-enhanced chemical vapor deposition               |
| PDFnc  | path deconfliction function                             |
| PSSA   | preliminary system safety assessment                    |
| RCLAW  | reversionary control law                                |
| RFMS   | reversionary flight management system                   |
| RGLAW  | reversionary guidance law                               |
| RM     | redundancy management                                   |
| RMPS   | reversionary mission planning system                    |
| ROR    | restricted operational region                           |
| RP     | rendezvous point                                        |
| RPV    | remotely piloted vehicle                                |
| RRSD   | risk/reward score database                              |
| RTA    | runtime assurance                                       |
| RWP    | reversionary waypoint                                   |
| SAA    | sense and avoid                                         |
| SIGINT | signal intelligence                                     |
| SoS    | systems-of-systems                                      |
| SSA    | system safety assessment                                |
| STINFO | scientific and technical information                    |
| UAS    | unmanned aircraft system                                |
| UAV    | uninhabited aerial vehicle                              |
| V&V    | verification and validation                             |
| VIPERS | very important person escort and response system        |

## Appendix A

### Simplex Survey

#### A.1 The Simplex Architecture – Review and Observations

A literature review of the simplex architecture was performed and this approach was analyzed for its relationship to the RTA architecture. Key papers on the simplex approach can be found in [Seto 1998], [Bak 2011], [Rivera 1996], [Sha 1995], [Sha 1998], [T. Johnson 2008], [Seto 1999], [Bak 2010]. Although similar to our past RTA approach, we are interested in the simplex approach because it allows the advanced or experimental controller to run outside the operational envelope of the baseline controller, and if a fault is discovered, a safety controller “drives” the system state back to where a baseline controller can run the system. Our past RTA approach limits operation of the advanced controller to the operational envelope of the baseline controller because no safety controller was considered.

The simplex approach was first used in computer network applications. The focus was on incorporating experimental code into a larger set of code and the effects and considerations of such a process. Much of the other work found in the literature present rather academic examples (e.g. controlling an inverted pendulum). Note that the design of the elements in the simplex architecture, namely the safety controller, is certainly more tractable for such simple academic applications.

The material presented in [Seto 1998] was found to be the most useful in describing the theoretical underpinnings of the safety controller and in presenting an application of the simplex architecture to a plasma-enhanced chemical vapor deposition (PECVD) system. However, we note that the PECVD application has a clearly defined, finite set of parameter limits that make developing a safety controller tractable. This has been noted in our RTA work in a related NASA Glenn project [Schierman 2014(a)]. Here, we developed an RTA system to monitor a turbofan engine that has an advanced controller. Our monitor observes the physical limit constraints on measurable parameters, such as fan speed and temperatures and pressures at various stages through the engine. For these types of applications, the design of the RTA system is more straightforward than for an aircraft’s controller, for example, where the safe regions to fly are complex functions of many states and parameters of the vehicle’s dynamics.

The simplex architecture from [Seto 1998] is presented in Figure A.1. Here, the user interface and update manager refers to swapping in/out experimental code modules as they become trusted through operational evidence. The experimental code is the advanced system that has not (or cannot) be fully certified. The baseline control module is a fully certified controller that will take over operation of the system if a fault is detected in the experimental code. The safety control module takes the system state from an operating point outside the operational region of the baseline controller to a safe operating point within the baseline controller’s operational region.

The following is largely reproduced from [Seto 1998] and describes the basic elements of the simplex architecture. Consider a physical system described by

$$\dot{x}(t) = f(x(t), u(t), t) \quad (\text{A.1})$$

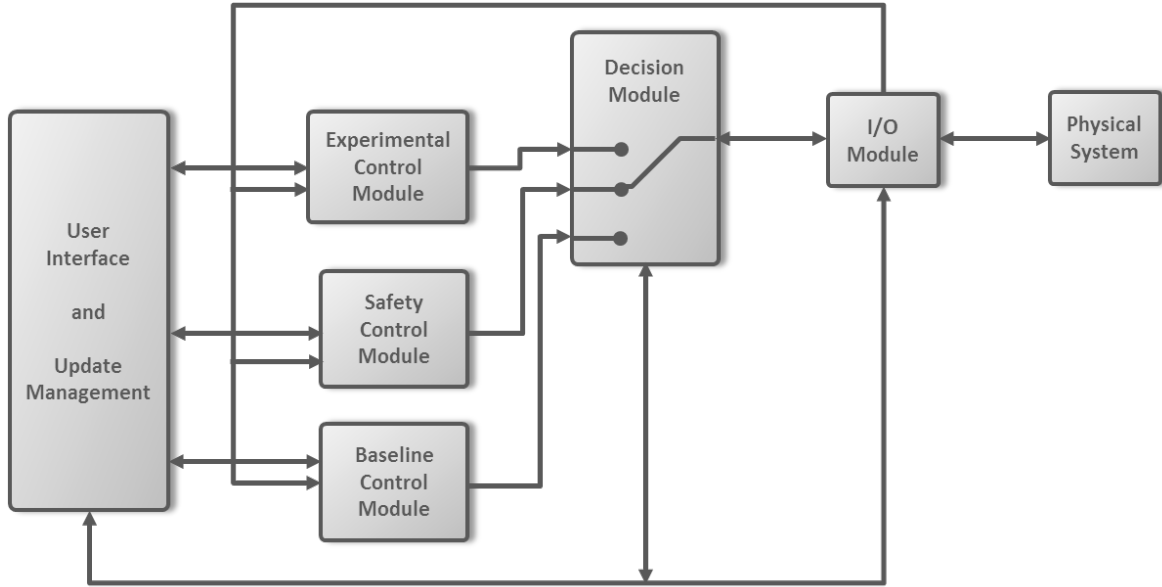
with state constraints given by

$$q_1(x) \leq 0, q_2(x) \leq 0, \dots, q_j(x) \leq 0, j < n \quad (\text{A.2})$$

and control constraints given by

$$p_1(u) \leq 0, p_2(u) \leq 0, \dots, p_r(u) \leq 0, r < m \quad (\text{A.3})$$

where  $x \in \mathbb{R}^n$  and  $u \in \mathbb{R}^m$  are the state and control inputs of the physical system, respectively.



**Figure A.1. The Simplex Architecture**

To provide protection against software faults, it is necessary to identify a region in the state space where the safety controller can control the system without violating the physical constraints. It is also necessary to identify the region within which the baseline controller can be applied without violating the physical constraints so that the control can be switched from the safety controller to the baseline controller at the appropriate time. These regions are defined as follows.

Let  $F \subseteq \mathbb{R}^n$  denote the set of admissible states, i.e., those satisfying the state constraints of (A.2),

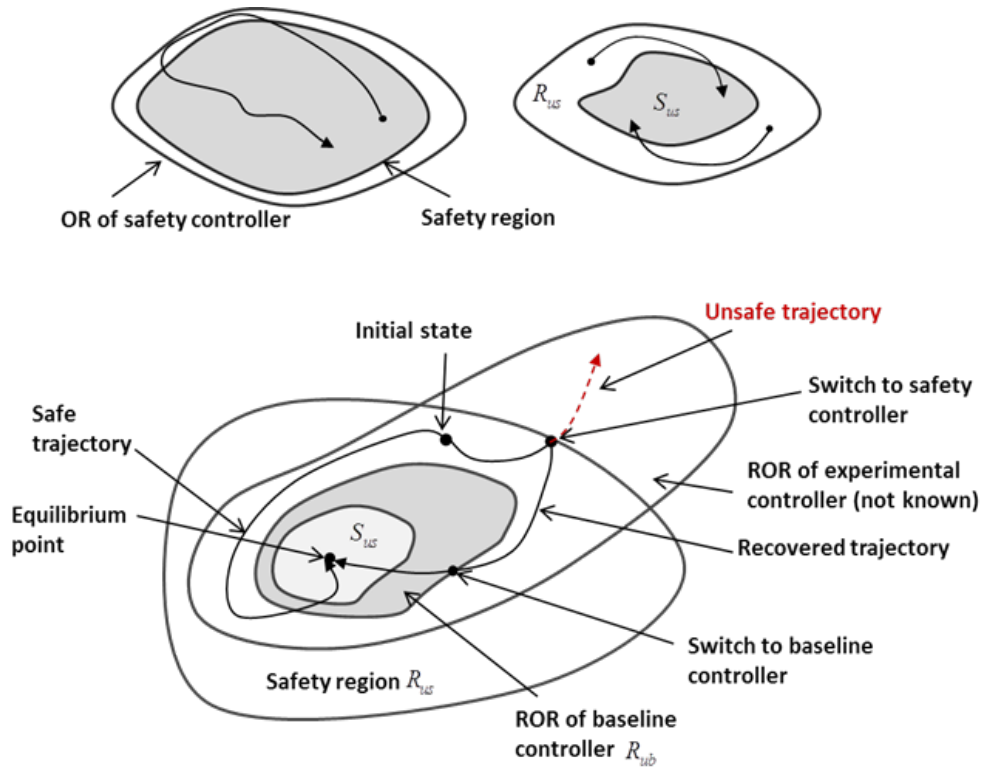
$$F = \{x : q_1(x) \leq 0, q_2(x) \leq 0, \dots, q_j(x) \leq 0\} \quad (\text{A.4})$$

and let  $\Omega \subseteq \mathbb{R}^m$  denote the set of admissible controls satisfying the control constraints of Equation A.3, i.e.,

$$\Omega = \{u : p_1(u) \leq 0, p_2(u) \leq 0, \dots, p_r(u) \leq 0\} \quad (\text{A.5})$$

We define the *operational region* (OR) for a given control law  $u$  taking values from  $\Omega$  to be a subset  $O_u \subseteq F$  such that if  $u$  is applied starting from any state in  $O_u$  then the resulting trajectory for the physical system remains in  $O_u$  and satisfies the control objective for  $u$ .

By characterizing the ORs for different control laws, we would like to establish a control switching logic based on what ORs the trajectory of the physical system fails in. The OR just defined, however, may not be sufficient for this purpose in digital control, where one sampling period delay of control is inevitable. To take the sampling period into account, we define a *restricted operational region* (ROR) as follows. Let  $T$  be the sampling period of the system and  $\phi_u(t_o, x_o, t)$  be the solution of Equation A.1 at  $t > t_o$  with  $u$  taking values from  $\Omega$  and  $(t_o, x_o)$  the initial condition. Then an ROR  $R_u$  of the control law  $u$  is defined as a subset of  $O_u$  and for all  $x \in R_u$ ,  $\phi_v(t_o, x_o, t_o + T) \in O_u \forall t_o > 0$  and  $\forall v \in \Omega$ . Using the above concepts, the philosophy of the simplex control switching logic can now be described and is illustrated in Figure A.2.



**Figure A.2. Simplex Control Switching Process**

The safety controller will be designed with the control objective of keeping the physical system from violating the physical constraints and to bring the system back to a state where the baseline controller can be used. Specifically, let  $u_s$  and  $u_b$  be the safety and baseline control laws, respectively. Then the *safety region* for the physical system is defined as the ROR of  $u_s$ , i.e. if all the trajectories of the physical system starting in  $R_{us}$  can be driven by  $u_s$  to a subset  $S_{us} \subset R_{us}$ , then the safety region is said to be *recoverable* to  $S_{us}$ . Since the objective of the

safety controller is to return the system to a state from which the baseline controller can take over, it is necessary that  $S_{us}$  be contained in the ROR for the baseline controller, that is,  $S_{us} \subset R_{ub}$ . The simplex switching strategy can then be described as follows: monitor the state of the physical system when the experimental controller is active; if the state reaches the boundary of  $R_{us}$ , switch to  $u_s$ ; then switch to  $u_b$  when the state reaches  $R_{ub}$ . If this strategy can be implemented, a given state of the physical system is said to be *safe* if it is inside  $R_{us}$ . Otherwise it is *unsafe*.

The rest of this appendix surveys existing work on the simplex architecture and similar architectures for run-time assurance.

## A.2 Survey of Simplex Approaches

This appendix surveys design methods for the Simplex architecture and similar architectures for RTA. A design method consists of a system architecture, a design for the decision module, and a method or algorithm for computing the switching condition used in the decision module, i.e., the condition that triggers a transfer of control from the advanced controller to the safety controller. The focus is on the method for computing the switching condition, since that is the most difficult part. The assumptions and limitations of each method are emphasized, to help determine their applicability to systems of interest in this project and other systems of interest to the Air Force.

The methods fall into two categories. The first category contains methods based on Lyapunov stability theory. These methods were developed by Lui Sha et al. ORTEGA is an extension of their design method to handle cold spares and digital controllers. NetSimplex is an extension of their design method to networked systems. The main advantage of these methods is that they are computationally inexpensive. The main disadvantage of these methods is that they have limited applicability; generally, they are limited to systems in which the plant (i.e., the controlled system) has linear dynamics and the safety controller is a linear controller that stabilizes the system at an equilibrium point.

The second category contains methods based on state-space exploration. These methods were developed by Stanley Bak, Sayan Mitra, et al. They are based on the observation that the set of unrecoverable states (and hence the switching condition) can be computed from the set of unsafe states using a reachability algorithm, i.e., an algorithm that computes the set of states reachable from a given set of initial states. A common feature of the reachability algorithms is that they *discretize* the state space. In other words, they divide the state space into regions, and they iteratively compute a set of regions whose union is the desired set of reachable states. There are many reachability algorithms, varying in the shape of the regions (e.g., boxes or regions bounded by polynomials), whether the partitioning of the state space into regions is pre-determined or adaptive, etc. The main advantage of these methods is their broad applicability: some reachability algorithms are applicable to general non-linear systems and to systems with multiple modes, each with different dynamics. The main disadvantage of these methods is that they are computationally expensive, especially for high-dimensional systems, because the number of regions may be large, and may grow exponentially in the number of dimensions. The required computational resources may be prohibitive in some cases, depending on the complexity of the system (especially the number of dimensions) and the required time horizon.

While the main impact is on the cost of the off-line algorithm used to compute the switching condition, there is also a potential impact on the run-time cost, i.e., the cost of evaluating the switching condition in the Simplex decision module. The most straightforward implementation of the decision condition is to store the set of regions along the boundary where the safety controller must take over, and check whether the current state is in one of them. If the set of regions is large, this is potentially expensive. The cost can be reduced by constructing a more compact approximate representation of the boundary.

We will also review Mitra et al.'s original and improved design methods, respectively, based on state-space exploration. We describe these methods in more detail than the methods based on Lyapunov stability theory, because they are more suitable for the systems of interest in this project, which have non-linear dynamics. The two methods differ mainly in the reachability algorithm. Indeed, any reachability algorithm can be used in their approach to computing the switching condition. This is evident from the formulation in [Bak 2011]. We have explored the use of other reachability tools for this problem, as described in the appendix on feasibility of computing switching conditions for keep-out zone avoidance and flight safety using formal methods tools.

### A.2.1 Sha et al.'s Design Method Based on Lyapunov Stability Theory

Sha et al.'s design method based on Lyapunov stability theory is described in [Seto 1998], [Seto 1999], [Sha 2001], with [Seto 1999] providing the most detailed exposition. [Seto 1998] presents the same framework and definitions as [Seto 1999] but does not describe algorithms for computing safety regions. It does present a specific design for the decision module for a Simplex architecture with three controllers (safety controller, baseline controller, and complex controller). The decision module is defined by a state machine that switches between controllers based on the state of the plant and which controllers are ready.

The method is formulated for continuous systems, not hybrid systems. In other words, the plant has a single mode with an associated dynamic model, and each controller has a single mode with an associated control law. The dynamic model of the plant has the form  $\dot{x} = f(x, u)$ , where  $x$  is the plant state, and  $u$  is the control input.

Sha et al. define a *safety region* of a (safety) controller as a *restricted operational region* (ROR) of the controller. The definitions of operational region (OR) and restricted operational region (ROR) are reproduced in Section A.1 of this report. The definition of ROR is expressed in terms of a solution  $\phi$  to the dynamic model for the system (comprising the controller and the plant). For complex systems, it is infeasible to obtain a closed-form solution to this dynamic model, so the definition of ROR does not directly provide an algorithm to compute the ROR. The definition quantifies over values of the control input and therefore does not require a model of the other (non-safety) controllers.

A ROR for a safety controller can be obtained without solving the dynamic model if a Lyapunov function for the system (comprising the safety controller and the plant) is available. A Lyapunov function is traditionally used to prove stability of a system, without solving the dynamic model. Informally, a *Lyapunov function* is a scalar function that (1) takes positive values everywhere in a given region (a stability region) except at the equilibrium points and (2) is non-increasing along every trajectory of the system. The stability region associated with a Lyapunov function



corresponds to an operational region (OR) of a controller (D. Seto 1999, Section 3.2) incorrectly states that a stability region corresponds to a safety region, i.e., an ROR). A safety region can be computed from an OR. [Seto 1999] does not describe how to do this, but [Bak 2010] does. Specifically, if bounds on the time derivative of the system state (i.e.,  $\dot{x}$ ) under the control of the complex controller, or under control of a non-deterministic controller whose possible behaviors include those of the complex controller, are available, these bounds, together with the Simplex control period  $\Delta$ , can be used in a straightforward way to compute the margin that separates an ROR from an OR.

Lyapunov functions for general non-linear systems cannot be obtained automatically. The paper gives an algorithm, based on Lyapunov stability theory, for computing a Lyapunov function and stability region for a linear controller of a linear time-invariant plant. A *linear time-invariant plant* is a plant whose dynamic model has the form  $\dot{x} = A x + B u$ , where  $A$  and  $B$  are matrices. A controller is *linear* if its control law has the form  $u = K x$ , where  $K$  is a matrix, called the *control gain matrix*. For such systems, the stability regions are characterized by linear matrix inequalities (LMIs), and the largest stability region can be computed by finding a solution to a set of LMIs that minimizes the logarithm of the determinant of the matrix. Such problems can be solved using the iterative algorithm in [Vandenberghe 1998]. In effect, the algorithm computes a Lyapunov function with a maximal-size stability region for the system. Generation of Lyapunov functions for certain classes of hybrid systems has been considered in [Oehlerking 2009].

### **Limitations of the Method**

The main limitation of this method is that most plants are not linear. The authors propose to address this limitation by linearizing the plant around an equilibrium point (see Section 3.2 of their paper) in the usual mathematical way, namely, by a first-order Taylor expansion. The higher-order terms are simply dropped, without regard for their sign and without a bound on their magnitude. However, this method of linearization does not ensure soundness of the safety-region calculation; in other words, the computed region might not actually be a safety region of the controller. To ensure soundness of the safety-region calculation, the linearization strategy must be designed to ensure that the approximation is "conservative" in some suitable sense. Details of the notion of "conservative approximation" in this context, and of how to obtain such approximations, are left for future work.

Another limitation of this method is that it applies only to systems with an equilibrium point and a safety controller that stabilizes the system at an equilibrium point. This limitation results from the use of Lyapunov stability theory. There are other similar techniques that can potentially be used to prove safety of a Simplex system without requiring stability (or existence of equilibrium points). Candidate techniques include *barrier certificates* [Prajna 2004], [Prajna 2007] *equational templates* [Sankaranarayanan 2008], and *differential invariants* [Platzer 2008], which serve a similar role in safety proofs as Lyapunov functions play in stability proofs. Using a technique designed for proving safety, instead of proving safety via a detour through stability, might make the verification easier, as well as more broadly applicable. Note that linearization of models of systems that do not have equilibrium points would require adopting a different way to choose linearization points.

Another limitation of the method is that it considers only continuous systems, not hybrid systems. In other words, it does not consider systems in which the plant has multiple modes,

each with a different dynamics, or the controller has multiple modes, each with a different control law.

### **A.2.2 ORTEGA: Extension of Sha et al.'s Design Method**

ORTEGA [Liu 2008] extends Sha et al.'s original work on Simplex to handle cold spares, controllers with different schedules, and digital controllers.

#### **Cold Spares**

In the original Simplex architecture, the high assurance controller (HAC) is a hot spare, while in ORTEGA, it is a cold spare. A *hot spare* is a backup system that runs continuously, even when the primary system is in control; this reduces the time needed to switch to the backup system. A *cold spare* is a backup system that is normally off and runs only when it needs to take over from the primary system. Thus, in ORTEGA, the HAC does not run in parallel with the high performance controller (HPC). Instead, when the switching module detects a problem, it stops running the HPC and starts running the HAC. The paper assumes that it takes one time period to do this, causing a delay before the HAC takes over. This delay needs to be taken into account when computing the switching condition.

The main benefit of making the HAC a cold spare is to lower the cost of the system by reducing the resource requirement. This design allows use of the Simplex architecture without the need to double the processing power of the system (in order to run the controllers in parallel). The disadvantages of this design are: (1) it forces the switching condition to be more conservative (because of the additional delay before the HAC can take over), and (2) it causes transitions from the HPC to the HAC to be less smooth. Making the HAC a cold spare may be desirable overall for applications such as industrial control, but is less likely to be acceptable in avionics applications, where smooth transitions are critical.

#### **Controllers with Different Schedules**

ORTEGA also extends the Simplex architecture and associated theory to allow the HAC and HPC to have (periodic) schedules with different periods and to have different execution times for their periodic tasks. In the original Simplex architecture, the two controllers were assumed to have the same scheduling requirements, for simplicity. The paper presents a scheduling analysis, based on rate-monotonic scheduling, to ensure that no deadlines are missed during steady states under both controllers and during changeovers between controllers.

#### **Digital Controllers**

ORTEGA includes a new method for computing the switching condition in the decision module. The method is based on Sha et al.'s design method based on Lyapunov stability theory for linear time-invariant systems [Seto 1999], described above. The method is modified to be suitable for digital controllers. Recall that, in [Seto 1999], the model takes into account that the decision module senses the system state and evaluates the switching condition only periodically at discrete times, but the dynamic model of the plant and controller is a continuous-time model, not a discrete-time model. In particular, the dynamic model assumes that the controller continuously senses the system state and instantaneously adjusts the control output. This assumption is reasonable for analog controllers but not digital controllers. The design method in ORTEGA is based on a discrete-time version of the dynamic equations, which is more realistic for digital controllers.

The paper shows that, for a linear time-invariant plant and discrete-time linear controller, the largest stability region and the associated Lyapunov function can be computed by finding a solution to a set of linear matrix inequalities that maximizes the logarithm of the determinant of the matrix.

If Simplex is used for a digital controller on a UAS, either the digital nature should be taken into account explicitly, or we can use a continuous-time model of the controller and argue that the control period of the digital controller is small enough that the continuous-time model is a sufficiently close approximation for the purpose of the Simplex safety argument. In particular, the approximation error should be quantified and bounded, and the safety proof should include a sufficient safety margin to allow for this approximation error.

### **Limitations of the Method**

ORTEGA's design method inherits the aforementioned limitations of the original method, namely the limitation to linear time-invariant systems, the limitation to systems with an equilibrium point and a safety controller that stabilizes the system at an equilibrium point, and the limitation to continuous (not hybrid) systems. In addition, work on ORTEGA ignores the critical issue of how to initialize the state of the HAC when switching to it. Presumably the authors believe that this requires application-specific techniques, and they do not have anything general to say about it.

### **A.2.3 NetSimplex: Extension of Sha et al.'s Method To Networked Systems**

NetSimplex ([Yao 2013]) extends the classic Simplex architecture (not ORTEGA) and associated theory to networked (i.e., distributed) systems. It considers distributed systems in which the sensor, controller, and actuator are at different nodes, and there are known bounds on the communication latency between them. Thus, there is a non-negligible delay between the time that a sensor reading is taken, and the time that the actuator produces the new value of the control input determined by the controller in response to that sensor reading. This delay includes the communication latency from the sensor to the controller, local processing time by the controller (this is relatively small), and the communication latency from the controller to the actuator. The paper assumes that an upper bound  $\tau$  on this delay is known.

In contrast, previous work on Simplex assumes that sensor sampling, controller response (i.e., computing a new control input in response to the new sensor value), and actuator actuation all occur at the same instant, at the beginning of each sampling period. Note: In ORTEGA [Liu 2008], this assumption is reflected in the discretization  $u(k) = -Kx(k)$  of the control law  $u(t) = -Kx(t)$  in Section IV.A.

Note that work on NetSimplex does not consider interactions between multiple instances of the Simplex architecture. NetSimplex adopts a more general model of the plant (i.e., the controlled system). In the classic Simplex architecture, the plant is assumed to be linear time-invariant (LTI). In other words, the plant's dynamics are described by an equation of the form  $\dot{x}(t) = Ax(t) + Bu(t)$ , where  $x$  is the system state,  $u$  is the control input,  $A$  is the system matrix, and  $B$  is the control matrix. Note that  $A$  and  $B$  are constant matrices. In NetSimplex, the plant is modeled as a linear parametric varying (LPV) system. This means that the system matrix and the control matrix are affine functions of time-varying parameter(s)  $\rho$ , hence are written as  $A(\rho)$

and  $B(\rho)$ . The parameter  $\rho$  is assumed to be measurable (in other words, its value cannot be predicted but can be measured) and to vary within a known interval. Also, the stability analysis requires a bound on the time derivative of  $\rho$ .

This generalization of the plant model requires a generalization of the controller model. In the classic work on Simplex, the HAC is assumed to be a linear feedback, with a control law of the form  $u(t) = Kx(t)$ , where  $K$  is the gain matrix. The rationale for using a linear feedback controller for the HAC in classic Simplex is that it generally achieves a large stability region, even though it might not minimize the control error. However, for a LPV plant, in general, one cannot find a single linear feedback controller with a large stability region for all values of  $\rho$ . Therefore, the NetSimplex architecture uses an interpolation gain-scheduling (IGS) controller as the HAC. This means that the HAC is actually a family of linear feedback controllers  $C_1, C_2, \dots$ , each with a gain matrix  $K_i$ . Each linear controller in the family is associated with a sub-interval  $[\underline{\rho}_i, \bar{\rho}_i]$  of the range of  $\rho$ . Control is switched to that controller when  $\rho$  falls into that interval. To ensure smooth control, control is not switched abruptly from controller  $C_i$  to controller  $C_{i+1}$  (or  $C_{i-1}$ ). Instead, the interval  $[\underline{\rho}_i, \bar{\rho}_i]$  associated with  $C_i$  is separated from the interval  $[\underline{\rho}_{i+1}, \bar{\rho}_{i+1}]$  associated with  $C_{i+1}$  by an interval  $[\underline{\rho}_i, \bar{\rho}_{i+1}]$ , within which an interpolated controller  $C_{i,i+1}(\rho)$  is used. The gain matrix  $K_{i,i+1}(\rho)$  of controller  $C_{i,i+1}(\rho)$  is a linear combination of  $K_i$  and  $K_{i+1}$ , with coefficients that depend on  $\rho$ .

The NetSimplex paper presents a method for computing stability regions for LPV systems with IGS controllers with network delays. The method is based on Sha et al.'s design method based on Lyapunov stability theory for linear time-invariant systems [Seto 1999], described above. Recall that [Seto 1999] shows, for linear time-invariant systems with linear feedback controllers, that computing a maximal safety region can be reduced to a Linear Matrix Inequality (LMI) problem, which essentially finds a Lyapunov function for the system. NetSimplex extends that method to compute a maximal safety region for a single linear feedback controller for a LPV system, taking delays into account. The main issue in the extension is taking network delays into account. The parameter  $\rho$  causes no trouble here, because it appears linearly in the equations. The resulting method can be used to compute a safety region for each controller  $C_i$  in the HAC.

The NetSimplex paper then shows how to combine these results by interpolation to obtain safety regions for each of the interpolation controllers  $C_{i,i+1}(\rho)$  and then for the entire HAC. This requires a bound on the time derivative of  $\rho$ . Informally, if  $\rho$  could change too quickly, it would not be sufficient to interpolate only between consecutive controllers  $C_i$  and  $C_{i+1}$ , because  $\rho$  could move from the interval associated with  $C_i$  to the interval associated with  $C_{i+2}$  before controller  $C_{i+1}$  had time to control the system. My intuition is that  $\rho$  must remain in the interval associated with  $C_i$  long enough for  $C_i$  to bring the system into the intersection of the stability regions of  $C_i$  and  $C_{i+1}$  (cf. Remark 5 in Section VI).

The NetSimplex paper describes how to compute switching conditions, based on the stability regions. In order to account for the delay between a decision to switch to the HAC, and the actuator receiving the first control input values from the HAC, at each decision time, the decision module uses the plant model to compute a "reach set" for the current state, i.e., an upper bound on the set of states reachable from the current state within time  $\tau$  under control of the HPC. If the reach set is not a subset of the stability region, then control is switched to the HAC. The

paper refers to other papers for details of the reach set computation. Note that this computation should be inexpensive, because it is performed on-line.

A model of a satellite attitude control system is used as a case study. The model is based on an example in the MATLAB LMI Toolbox manual. The controller controls the satellite's yaw angles. The parameters in the dynamic model are torque constant and viscous damping. The HPC is a copy of the HAC with an injected bug (similar to divide-by-zero) that causes the HPC to produce large values for the control input partway through the execution. Two other potential applications, suggested by Xue Liu, are robotic surgery guided by a remote surgeon (e.g., the patient and robot are on the battlefield; the surgeon is not), and remotely-guided disaster recovery robots.

### **Limitations of the Method**

The NetSimplex design method is limited to plants with linear parametric varying (LPV) dynamics and an interpolation gain-scheduling (IGS) controller, as described above. Thus, it is not applicable to non-linear plants in general, but it is applicable to a non-linear plant if the state components with non-linear behavior can be included in the parameter  $\rho$ , leaving only the components with linear behavior, for fixed  $\rho$ , to be modeled as the system state  $x$ . Similar to Sha et al.'s original design method, the NetSimplex design method is limited to systems with equilibrium points and safety controllers that stabilize the system at equilibrium points, and it is limited to continuous (not hybrid) systems.

### **A.2.4 Mitra et al.'s Original Design Method Based On State-Space Exploration**

[Bak 2010] describes an algorithm for computing the switching condition for a decision module in a Simplex architecture involving two controllers: a safety controller and a complex controller. The method applies to hybrid systems, i.e., systems with multiple modes, each with a different dynamics. We present the model of hybrid systems and then the algorithm. The paper also presents a case study of using Simplex to ensure that an off-road vehicle, such as a tractor, does not roll over, using a safety controller that reduces the steering angle (i.e., straightens the conceptual steering wheel) and velocity.

### **System Model**

Hybrid automata are widely, with minor variations, in the research community to model cyber-physical systems. This paper uses the following definition.

A *hybrid automaton* (HA) consists of:

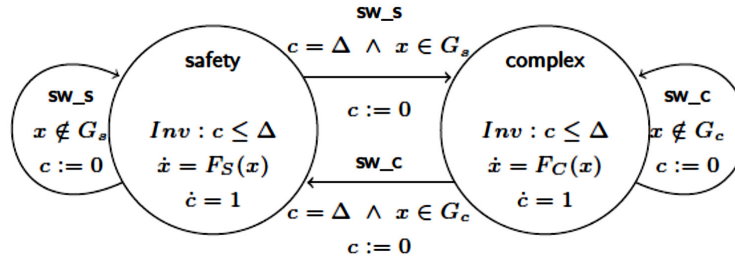
- $V$ : a set of variables. The variables may range over discrete or continuous domains. A *state* is a mapping from  $V$  to values.
- $\Theta$ : a set of initial states
- $A$ : a set of "actions". Each action has a *guard condition*, which is a condition that must hold for the action to occur, and a *reset map*, which is an assignment of constant values to specified variables.
- $D$ : a set of transitions. Each transition has the form  $(v, a, v')$  where  $v$  and  $v'$  are states, and  $a$  is an action.

- $T$ : a set of trajectories. A *trajectory* is a map from a time interval  $[0, t]$  to states, describing the continuous evolution of the system (in between transitions). A trajectory is represented by a set of differential equations for the continuous variables and an *invariant*, which is a set of inequalities that must hold during the trajectory. The invariant limits the duration of the trajectory: the trajectory must end before the invariant is violated.

The system architecture is represented by a generic HA with three variables:

- a variable named "mode" whose values correspond to the choice of controller: mode=safety or mode=complex.
- a clock variable  $c$  that drives the periodic check of the switching condition.
- application-specific real-valued variables  $x_1, \dots, x_n$ . Let  $x$  denote this vector of variables.

The HA is depicted in the following diagram. Note that  $sw\_s$ , etc., are names for the actions, and the formulas labeled "Inv" are the invariants.



**Figure A.3. Hybrid Automaton**

To simplify the model, the safety controller and complex controller are not distinguished from the plant. The combined dynamics of the safety controller and the plant is characterized by  $\dot{x} = F_S(x)$  (Note: this is a vector of differential equations, since  $x$  is a vector of real-valued variables). The combined dynamics of the complex controller and the plant is characterized by the differential equation  $\dot{x} = F_C(x)$ .  $F_S$  and  $F_C$  are required to be continuous, are not required to be linear, and are subject to some restrictions, described below.

The HA contains two *mode guards* (i.e., switching conditions)  $G_s$  and  $G_c$ . The value of mode changes from safety to complex if, at the pre-state of the transition, the valuation of  $x$  is in  $G_s$ . Similarly, mode changes from complex to safety if  $x$  is in  $G_c$  in the pre-state. (This description is copied from Section III of the paper, after fixing a confusing typo:  $G_s$  and  $G_c$  are swapped.)

The clock variable  $c$  is used to trigger a transition every  $\Delta$  time units, where  $\Delta$  is the Simplex decision period. Each transition evaluates the mode guards and, depending on the result, either stays with the current controller or switches to the other controller.

The mode guards  $G_s$  and  $G_c$  are initially unknown. The main contribution of the paper is an algorithm for computing the mode guards, from the differential equations and a specification of the set  $U$  of unsafe states.

For simplicity, the algorithm presented in the paper assumes that, in each of the two Simplex modes (i.e., safety mode and complex mode), the plant and controller are continuous systems (not hybrid systems). The paper states that the algorithm can be generalized to apply to systems where the behavior of the system in each Simplex mode is described by a hybrid automaton.

Let  $\text{Simplex}_{mode}$  (where the subscript  $mode$  is a meta-variable whose value is “safety” or “complex”) denote the HA representing the behavior of the system locked in the indicated mode. Specifically, the mode variable and the switching logic are eliminated. Thus,  $\text{Simplex}_{mode}$  has variables  $\{x_1, \dots, x_n, c\}$ , has trajectories defined by the appropriate differential equations, and has no transitions.

### **Overview of the Algorithm**

**Step 1.** Construct a discrete abstraction  $\text{AbsSimplex}_{mode}$  of each HA  $\text{Simplex}_{mode}$ , by quantizing (i.e., discretizing) the state space, i.e., partitioning the state space into fixed-size rectangular regions. This approach is based on the classical notion of *abstract interpretation* in software analysis. The following restriction is imposed to ensure that the abstract state space is finite:

- R1: There are known bounds on the possible values of each real-valued state variable  $x_i$ .

A *discrete abstraction* of a HA  $A$  is a labeled transition system (finite state machine)  $\text{Abs}A$  with:

- a finite set of states, each corresponding to a region in the state space of  $A$ . Functions used to convert between concrete states and abstract states are:

$\text{QUANTIZE}(v)$  maps a concrete state  $v$  to the corresponding abstract state

$\text{QUANTIZE-SET}(s)$  maps a set of concrete states (i.e., a subset of  $\mathbb{R}^n$ ) to the corresponding set of abstract states. Note: there is a typo in Section IV.B of the paper: the type of  $\text{QUANTIZE-SET}(s)$  should be  $2^{\mathbb{R}^n} \rightarrow 2^{\text{val}(Y)}$ , not  $\mathbb{R}^n \rightarrow 2^{\text{val}(Y)}$ .

$\text{QUANTIZE}^{-1}(y)$  maps an abstract state to the corresponding region of concrete states, represented as a Cartesian product of left-closed, right-open intervals.

- transitions corresponding to transitions of  $A$
- transitions corresponding to trajectories of  $A$

The abstract transitions must represent the possible behaviors of the HA; specifically, they must satisfy the requirement:

For all abstract states  $y_1$  and  $y_2$ , if there is a concrete state  $v_1$  in  $\text{QUANTIZE}^{-1}(y_1)$  and a concrete state  $v_2$  in  $\text{QUANTIZE}^{-1}(y_2)$  such that  $A$  has a transition or trajectory from  $v_1$  to  $v_2$ , then  $\text{Abs}A$  has a transition from abstract state  $y_1$  to abstract state  $y_2$ .

Note that this is weaker than the requirement in Definition 3 (the definition of discrete abstraction), which uses “iff” instead of “if”. The labeled transition system computed by the algorithm does not satisfy the stronger condition in Definition 3, because the use of bounds on derivatives, instead of exact solutions to differential equations, in the construction implies that the abstraction might contain some unnecessary transitions (the paper mentions this in the last paragraph of Section IV.C).

**Step 2.** Compute the mode guards by analyzing the behavior of the discrete abstraction to compute the following sets of states.

$\text{AbsUnsafe}$  = unsafe abstract states, i.e., states of  $\text{AbsSimplex}_{\text{safety}}$  that correspond to regions that overlap with the set  $U$  of unsafe states. Formally,  $\text{AbsUnsafe} = \text{QUANTIZE-SET}(U)$ . Note: there is an error in the displayed formula in the first paragraph of Section IV.D of the paper:  $U$  should be replaced with  $\text{QUANTIZE-SET}(U)$ .

$\text{AbsLeadToUnsafe}$  = abstract states from which a state in  $\text{AbsUnsafe}$  is reachable via transitions of  $\text{AbsSimplex}_{\text{safety}}$ . In other words, if the system enters one of these abstract states while under the control of the safety controller, it can hit an unsafe abstract state in the future. In other words, these are states from which the safety controller cannot guarantee recovery.

$\text{AbsG}_c$  = abstract states from which a state in  $\text{AbsLeadToUnsafe}$  is reachable via one transition of  $\text{AbsSimplex}_{\text{complex}}$ . In other words, if the system enters one of these abstract states while under the control of the complex controller, it can enter a state from which the safety controller cannot guarantee recovery, even if we switch to the safety controller at the next time step.

$G_c$  = set of regions in the state space corresponding to abstract states in  $\text{AbsG}_c$ . Note that each region is one of the partitions defined during discretization, hence is a Cartesian product of intervals. Formally,  $G_c = \text{QUANTIZE}^{-1}(\text{AbsG}_c)$ .

The mode guard  $G_s$  that triggers switching from the safety controller to the complex controller can be taken to be the complement of  $G_c$  or a subset thereof (in order to prevent frequent switching). The paper does not discuss how to choose an appropriate subset.

#### **Details of the Algorithm: Computing Abstract Transitions**

The computations in Step 2 are easy, because they work mainly with the discrete abstraction. The difficult step is construction of the abstract transition system in Step 1, specifically, determining the abstract system's transitions corresponding to trajectories of the HA, because this involves the differential equations of the HA. Two restrictions are imposed on the HA to simplify this step:

- R2: For any rectangular region  $R$ , we can compute upper and lower bounds on the values of  $F_S$  and  $F_C$  within region  $R$ . The paper does not discuss how to compute these bounds.
- R3: The dependency graph of the continuous variables induced by the differential equations  $\dot{x} = F_S(x)$  is acyclic, save for self-dependencies. The same requirement also applies to  $\dot{x} = F_C(x)$ .

Restriction R3 implies that we can sort the variables in dependency order. For concreteness, we assume  $x_1, \dots, x_n$  are already sorted in this order. For each abstract state  $y$ , for each variable  $x_i$ , the algorithm computes the range  $r_i$  of values that  $x_i$  can reach from concrete states in  $\text{QUANTIZE}^{-1}(y)$  within time  $\Delta$ . Range  $r_i$  is determined using the bound on  $\dot{x}_i$  obtained from Restriction R2. Restriction R3 implies that this bound on  $\dot{x}_i$  can be computed from bounds on the values of  $x_1, \dots, x_{i-1}$  and bounds on the value of  $x_i$ . For the bounds on  $x_1, \dots, x_{i-1}$ , we use the previously computed ranges  $r_1, \dots, r_{i-1}$ . For the bounds on  $x_i$ , we initially use  $\text{QUANTIZE}^{-1}_i(y)$ , which denotes the range of values of  $x_i$  in  $\text{QUANTIZE}^{-1}(y)$ . This leads to an initial estimate of  $r_i$ . If  $r_i$  is within  $\text{QUANTIZE}^{-1}_i(y)$ , we are finished. If not, we compute how long it takes  $x_i$  to reach



the boundary between  $\text{QUANTIZE}^{-1}(y)$  and some neighboring abstract state  $y'$ , and then perform a similar calculation, using bounds on  $\dot{x}_i$  computed with  $\text{QUANTIZE}^{-1}_i(y')$  instead of  $\text{QUANTIZE}^{-1}_i(y)$ , to determine the range of values that  $x_i$  can reach in  $y'$  in the time remaining in the time step, and we use that range as an updated estimate of  $r_i$ . This process is repeated until it reaches an abstract state  $y'$  such that  $x_i$  remains within  $\text{QUANTIZE}^{-1}_i(y')$  for the remainder of the time step.

The Cartesian product  $r_1 \times \dots \times r_n$  of these ranges is an over-approximation of the set of concrete states reachable from states in  $\text{QUANTIZE}^{-1}(y)$  within time  $\Delta$ . There are transitions from  $y$  to abstract states corresponding to those concrete states, i.e., to abstract states in  $\text{QUANTIZE-SET}(r_1 \times \dots \times r_n)$ .

Note: The paper describes this construction slightly differently: it first uses the intervals  $r_i$  to compute a set of reachable values for each abstract variable  $y_i$ , and then takes the Cartesian product of those sets of abstract values, in order to produce the set of abstract states reachable from  $y$  in one step. That construction is equivalent to the one described above but requires some additional notation to write down neatly.

### **Limitations of the Method**

The method can handle non-linear hybrid systems that satisfy the restrictions R1 to R3 presented above. R1 (bounds on values of variables) is probably reasonable in most cases when considering time-bounded reachability but questionable for unbounded reachability. R2 (bounds on derivatives in a given box) is probably reasonable in most cases. R3 (acyclic dependencies between variables, ignoring self-loops) is reasonable only for systems that do not use feedback control. However, feedback control is common. For example, a controller for a UAS might use a feedback loop for thrust to maintain constant velocity; this induces a cyclic dependency between  $T$  and  $V$ .

In addition to these explicit limitations, there is an implicit limitation to systems for which the reachability algorithm provides sufficiently accurate results with reasonable computational resources. The granularity of the partition (i.e., the size of each region) used in the discretization of the state space is a tunable parameter that controls the trade-off between cost and accuracy. Finer granularity increases the cost (i.e., the time and memory used by the algorithm) and accuracy. The consequence of reduced accuracy is a more “conservative” switching condition, which may cause the system switches to the safety controller unnecessarily. Despite the flexibility to adjust the granularity, the performance of this relatively simple reachability algorithm is intrinsically limited by the fact that it uses a completely pre-determined discretization (i.e., set of regions) and the fact that it abstracts all of the trajectories into abstract transitions based on that discretization (in Step 1) before the reachability calculation (in Step 2). Mitra et al.’s improved design method, discussed next, relaxes the explicit restrictions and improves the performance by computing abstract transitions in a more adaptive way, during the reachability calculation.

### **A.2.5 Mitra et al.’s Improved Design Method Based on State-Space Exploration**

The most important contribution of [Bak 2011] is that it demonstrates in a general way that computation of switching conditions can be reduced to reachability problems for hybrid automata, and hence any reachability algorithm can be used. In addition, it presents a specific

reachability algorithm for hybrid automata. Compared to their original algorithm [Bak 2010], described in the previous section, the new technique is applicable to a broader class of hybrid systems and more accurate. S. Bak subsequently developed an improved reachability algorithm for hybrid automata. The improved algorithm, based on face lifting [Dang 1998], is not documented in publications but is incorporated in the HyCreate2 tool for verification of hybrid automata. Some other verification tools for hybrid automata, and the reachability algorithms they implement, are described briefly in the appendix on Feasibility of Computing Switching Condition for Keep-Out Zone Avoidance and Flight Safety Using Formal Methods Tools.

As a case study, [Bak 2011] considers a controller for an autonomous land vehicle required to visit a sequence of waypoints while remaining within a fixed distance of the line connecting successive waypoints. The safety controller simply stops the vehicle as quickly as possible.

### **Overview of the Algorithm**

Let  $\text{BackReach}(\text{target}, \text{sys})$  denote the set of states that lead system  $\text{sys}$  to a state in  $\text{target}$ ; in other words, state  $s$  is in the set if, when executing  $\text{sys}$  starting in  $s$ , some state in  $\text{target}$  is eventually encountered. We say that these states are *backwards reachable* from target. Let  $\text{BackReach}_{\leq \delta}(\text{target}, \text{sys})$  denote the set of states that lead system  $\text{sys}$  to a state in  $\text{target}$  in time at most  $\delta$ ; this is called *time-bounded backwards reachability*.

Let  $SC$  denote the composition of the safety controller and the plant (i.e., the controlled system). Let  $CC$  denote the composition of the complex controller and the plant. Let  $U$  be the set of unsafe states.  $\text{BackReach}(U, SC)$  is the set of unrecoverable states: if the system ever gets into one of these states, the safety controller might not be able to save the system (i.e., prevent it from entering an unsafe state). Let  $G = \text{BackReach}_{\leq \delta}(\text{BackReach}(U, SC), CC)$ , where  $\delta$  is the Simplex decision period.  $G$  is the set of states such that the system, under control of the complex controller, might enter an unrecoverable state before the next decision time.

Since we lack an exact model of the complex controller, we use an overapproximation  $CC'$  of  $CC$ . Note that the controllers (and the plant) may be non-deterministic, i.e., have multiple possible behaviors in a given state. In the absence of any certified information about the behavior of the complex controller, a completely non-deterministic model can be used for  $CC'$ , i.e., a model that, in every state, might produce every possible value of the plant's control inputs.

Also, the algorithms for computing backwards reachability compute overapproximations of the desired sets of states. Therefore, the method actually computes a set  $G' \supseteq \text{BackReach}_{\leq \delta}(\text{BackReach}(U, SC), CC')$ .  $G'$  is used as the switching condition; in other words, the decision module switches to the safety controller if the state is in  $G'$ . Note that  $G' \supseteq G$ , so this is a conservative (i.e., safe) approximation.

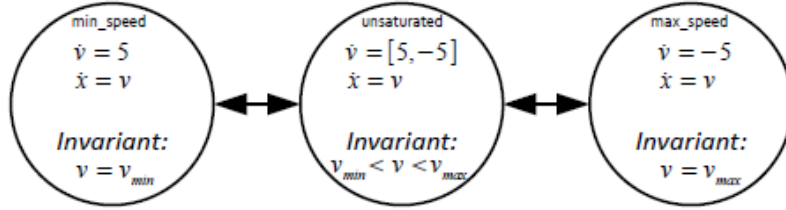
Thus, the problem of computing a switching condition has been reduced to computing overapproximations of backwards reachability and time-bounded backwards reachability. We describe the system model and then sketch the algorithms.

### **System Model**

The systems  $SC$  and  $CC'$  are modeled as hybrid automata. A *hybrid automaton* (HA) consists of:

- $V$ : a set of (continuous) variables. A *continuous state* is a mapping from  $V$  to values.
- $L$ : a set of locations, and an initial location  $l_0$ . Locations are the discrete states of the HA. Each location has an *invariant*, which is a condition on the values of the continuous variables). The system can remain in a location only while the location's invariant is satisfied. A *state* consists of a location and a continuous state.
- $S$ : a set of (valid) continuous states, and a set  $\Theta$  of initial continuous states.
- $D$ : a set of transitions. Each transition has a starting location, a guard condition, a reset map (i.e., a set of non-deterministic assignments to variables), and a target location.
- $T$ : a set of trajectories, which define the evolution of the continuous state variables while the system is in a given location. Trajectories are represented by a differential inclusions for variables in  $V$ . A *differential inclusion* consists of inequalities that provide upper and lower bounds on the derivative of a variable. For example, the differential inclusion  $\dot{x} = [-1, 1]$ , which is short-hand for  $-1 \leq \dot{x} \leq 1$ , allows all trajectories in which  $\dot{x}$  is between -1 and 1. Note that a differential equation is a special case of a differential inclusion in which the upper and lower bounds are equal. Differential inclusions can express non-deterministic behavior. This is important in the context of Simple, because it allows non-deterministic models of complex controllers, whose behavior may be either unknown or known but uncertified.

An example hybrid automaton appears below. Each circle represents a location, with the name of the location at the top.



**Figure A.4. An Example Hybrid Automaton**

### **Assumptions**

The reachability algorithm makes two assumptions about (i.e., imposes two requirements on) the hybrid automaton.

- Assumption 1: For any hyperrectangle  $H$ , and any continuous variable  $x$ , we can compute upper and lower bounds on the value of  $\dot{x}$  in  $H$ . Let  $\text{dbMin}(x, H)$  and  $\text{dbMax}(x, H)$  denote these bounds. These bounds are obtained by analyzing the differential inclusions describing the trajectories. The paper does not discuss how to do that analysis. A general approach is to find the extremal points of  $\dot{x}$  in  $H$  (i.e., points in  $H$  where  $\ddot{x} = 0$ ), evaluating  $\dot{x}$  at those points, and taking the smallest and largest of the resulting values. Note that the differential inclusions are used *only* to obtain  $\text{dbMin}$  and  $\text{dbMax}$ ; the reachability algorithm does not directly use the differential inclusions. Thus, the differential inclusions can be arbitrarily complicated and non-linear, provided these bounds can be obtained.

- Assumption 2: The dependency graph of the continuous variables induced by the differential inclusions is acyclic, save for self-dependencies. The *dependency graph* contains an edge from  $y$  to  $x$  if the differential equation for  $\dot{x}$  uses  $y$ . These are called *explicit dependencies*. Although this appears to be the same as restriction R3 in (S. Bask 2010), the paper says “This restriction is more relaxed than in our previous work, where the combined explicit / implicit derivative dependency graph was required to be acyclic, except for self-loops.” This implies that the dependency graph used in assumption R3 in [Bak 2010] includes implicit dependency edges, although [Bak 2010] does not mention this. The dependency graph contains an *implicit dependency edge* from  $y$  to  $x$  if a change in the value of  $y$  can cause a transition that changes the trajectory for  $x$  (i.e., a transition to a location with a different equation for  $\dot{x}$ ).

### **Algorithms for Overapproximating Reachability and Time-Bounded Reachability**

The algorithm in this paper, like the algorithm in [Bak 2010], is based on quantizing the state space, i.e., partitioning it into fixed-size hyperrectangles. Let  $q_i$  denote the quantum for  $x_i$ ; informally,  $q_i$  is the granularity used in calculations involving  $x_i$ . In both algorithms, the quanta  $q_i$  and a time quantum  $\delta$  are used to discretize the calculation and to help control the trade-off between cost and accuracy. However, there are important differences between the algorithms. The algorithm in [Bak 2010] first computes a discrete abstract system, by quantizing the states and computing a transition relation between the abstract states, and then performs the reachability calculation on the discrete abstract system. Each abstract state corresponds to a hyperrectangle (in the concrete state space) aligned on quantum boundaries (i.e., the edges in dimension  $x_i$  are on multiples of  $q_i$ ); let's call these *grid cells*. Consequently, the algorithm in (S. Bak 2010) is limited to computing sets of states that are unions of grid cells. In contrast, the new algorithm does not compute an abstract system. The new algorithm performs the reachability calculation directly with hyperrectangles that are not necessarily aligned on quantum boundaries. This can improve accuracy.

Forward reachability and backwards reachability are computationally equivalent (in other words, algorithms for one of these problems can easily be converted to an algorithm for the other), and forward reachability is more intuitive, so the paper presents algorithms for computing overapproximations of forward reachability and time-bounded forward reachability, instead of algorithms for computing overapproximations of backwards reachability and time-bounded backwards reachability.

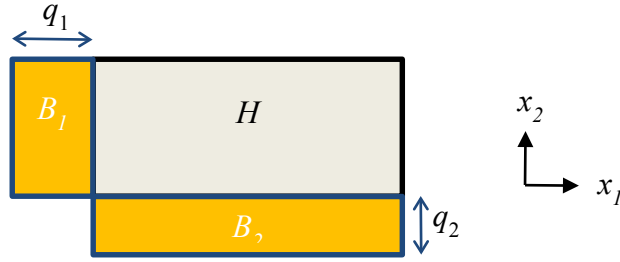
Let  $\text{Reach}(I)$  denote the set of states reachable from states in  $I$ , where  $I$  is a set of continuous states. Let  $\text{Reach}_{\leq \delta}(I)$  denote the set of states reachable from states in  $I$  in time at most  $\delta$ . High-level pseudocode for the algorithm for computing  $\text{Reach}(I)$  appears below. The hybrid automaton, the quanta  $q_i$ , and the time quantum  $\delta$  are implicit arguments to the algorithm. Each iteration of the while-loop advances time by  $\delta$ , so the algorithm for  $\text{Reach}_{\leq \delta}(I)$  is the same except that it always terminates after exactly one iteration of the while-loop.

```

Result = set of grid cells that overlap with I
E = array containing grid cells that overlap with I
oldResult = emptyset % any value different than Result is fine.
while Result ≠ oldResult
 oldResult = Result
 for i = 1 to length(E)
 E' = a hyperrectangle containing all states to which states in E[i] can evolve after time
 δ
 H' = the smallest hyperrectangle that encloses E[i] and E'
 E[i] = E'
 Result = Result ∪ H'
 end for
end while

```

The heart of the algorithm is the subroutine that computes a hyperrectangle containing all states to which the states in a given hyperrectangle  $H$  can evolve after time  $\delta$  works as follows. It considers (i.e., determines the range of possible values of) one variable at a time. Variables are considered in dependency order, based on the explicit dependency graph described in Assumption 2. Implicit dependencies are taken into account by re-considering variables when necessary; specifically, after considering each variable  $x_i$ , if the range of possible values of  $x_i$  increased enough to allow the hybrid automaton to transition to a location with a different trajectory for a variable  $x_j$  that has already been considered, then  $x_j$  is re-considered. To determine the lower bound of the range of possible values of variable  $x_i$  after time  $\delta$ , consider the hyperrectangle  $B_i$  that is adjacent to  $H$  in the negative  $x_i$  direction and whose width in the  $x_i$  dimension is  $q_i$ . This is illustrated below for the 2-dimensional case, assuming  $x_1$  is horizontal and  $x_2$  is vertical. Use dbMin to determine the minimum derivative (i.e., rate of change) of  $x_i$  in  $B_i$ . Let  $x_i$  decrease at that rate, starting from the edge of  $H$ , until either the given time  $\delta$  runs out or  $x_i$  reaches the far edge of  $B_i$  in some amount of time  $t$  with  $t < \delta$ . In the former case, use the final value of  $x_i$  as the desired lower bound. In the latter case, recursively use the same method to determine the lower bound for  $x_i$  in the hyperrectangle containing all states to which states in hyperrectangle  $B_i$  can evolve after time  $\delta - t$ , and use the result of that calculation as the desired lower bound. Note that this has the effect of making the timestep adaptive, in the sense that the timestep is effectively the smaller of  $\delta$  and the time needed for the value of  $x_i$  to change by  $q_i$ . For additional details of the algorithm (e.g., related to reversing direction, i.e., when dbMin has a different sign in the recursive call), see the paper. The upper bound of the range of possible values of  $x_i$  is computed similarly, using a hyperrectangle of width  $q_i$  that is adjacent to  $H$  in the positive  $x_i$  direction, and using dbMax instead of dbMin.



**Figure A.5. A Hyperrectangle**

### **Increasing the Accuracy**

The paper describes 3 strategies for increasing the accuracy. (1) Split the quanta for some variables into smaller quanta. (2) Split some hyperrectangles in  $E$  in some iterations, i.e., replace a hyperrectangle  $H$  with hyperrectangles  $H_1, \dots, H_n$  that partition  $H$ . (3) Split the time quantum into smaller time quanta. All of these strategies cause dbMin and dbMax to be called on smaller hyperrectangles, yielding tighter bounds on the derivatives within each hyperrectangle, thereby increasing the accuracy of the computation.

The effectiveness of the strategies is demonstrated by applying them a manually specified number of times to the running example (a simple controller for the acceleration of a vehicle), and comparing the computed overapproximation of the reachable states to the set of states reachable in simulations.

### **Shortcomings of the Algorithm**

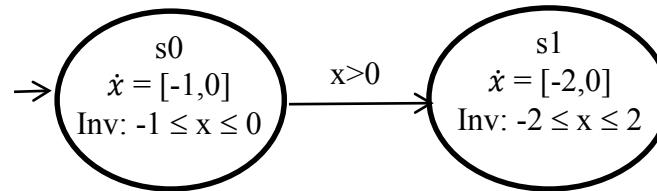
The *error* in the algorithm's output is the difference between the computed overapproximation of the reach set and the actual reach set. Theorem 1 states that, under some assumptions, applying the three accuracy-increasing strategies sufficiently many times will reduce the error below any given positive number  $\epsilon$ . An extended version of the paper, available from the authors by request, contains a proof of Theorem 1. This is encouraging, but three shortcomings should be noted.

(1) The paper does not discuss how to determine how many times each strategy needs to be applied to reduce the error below a given bound  $\epsilon$ . Generally, the paper does not discuss how to control the application of the accuracy-increasing strategies. For example, for strategy 2, the paper does not discuss how to decide which hyperrectangles to split in each iteration. One strategy is, after an initial calculation of the switching condition, to split hyperrectangles on the boundary of the switching condition.

(2) The algorithm might diverge or have larger error in cases where it could easily terminate or reduce the error by taking invariants associated with locations into account. Note that the invariants are not mentioned explicitly in the description of the algorithm. Implicitly, invariants are used in the `getLocations( $H$ )` function, which takes a hyperrectangle  $H$  as input and returns the set of locations in which the automaton may be when the continuous state is in  $H$ . The paper does not describe how to compute this function, but presumably this function returns the set of locations whose invariant overlaps with  $H$  (in other words, the locations such that  $H$  contains a state that satisfies the location's invariant). The algorithm does not take invariants into account

when computing the boundaries of hyperrectangles. For an example where this leads to divergence, consider the hybrid automaton having a single location with trajectory  $\dot{x} = [-1,0]$  and invariant  $-1 \leq x \leq 0$ . The initial value of  $x$  is 0. According to the pseudo-code in the paper, the algorithm diverges, exploring increasingly large negative values of  $x$ . This can be fixed by modifying ComputeDeltaReach to intersect  $\mathcal{E}$  and  $\mathcal{H}$  with union of the invariants of the locations in  $\mathcal{L}$  before returning them.

(3) The theorem is stated in a way that hides a limitation of the accuracy of the algorithm. The theorem and algorithm do not consider the possibility of increasing accuracy by keeping track of the set of reachable location-hyperrectangle pairs, instead of just the set of reachable hyperrectangles. This is related to the fact that the algorithm does not fully take the guards and reset maps on transitions into account. (My understanding is that the algorithm takes guards into account only when determining implicit dependencies, and the algorithm never takes reset maps into account.) The accuracy-increasing strategies cannot overcome this limitation. To see how this reduces the accuracy of the algorithm, consider the following hybrid automaton, where the initial value of  $x$  is 0:



**Figure A.6. Example Hybrid Automaton with Initial  $x = 0$**

Because the algorithm does not consider guards when determining reachability of locations, it does not realize that location  $s1$  is unreachable, and the set  $\mathcal{L}$  of locations contains  $s0$  and  $s1$ . Even with the modification described in item (2) above to intersect the reachable hyperrectangles with the invariants of the locations in  $\mathcal{L}$ , the algorithm concludes that  $[-2,0]$  is reachable, whereas actually only  $[-1,0]$  is reachable. Stanley Bak agreed that these are shortcomings of the algorithm presented in the paper. He also said that the paper presents a simplified form of the algorithm that they implemented, and issue (2) is addressed in their implementation.

### **Limitations**

This algorithm is a significant improvement over the algorithm in [Bak 2010]. Aside from the specific shortcomings mentioned above, it has some limitations due to the assumptions it makes. Assumption 1 (bounds on derivatives in a given box) is probably reasonable in most cases. Assumption 2 (acyclic dependencies between variables, ignoring self-loops) is reasonable only for systems that do not use feedback control. However, feedback control is common. For example, a controller for a UAS might use a feedback loop for thrust to maintain constant velocity; this induces a cyclic dependency between  $T$  and  $V$ .

As mentioned in the beginning of this section, this algorithm is superseded by a reachability algorithm based on face lifting [Dang 1998], developed by Stanley Bak and implemented in the HyCreate2 tool. This algorithm does not require Assumption 2 and addresses issues (2) and (3) mentioned above.

## Appendix B

### Adverse Emergent Behavior and Flocking Dynamics

Before developing a detailed mission management level challenge problem, we investigated abstract system-of-systems that exhibit adverse emergent behavior so that we can potentially explore RTA requirements to detect and properly address such behavior. An RTA system should be able to recognize the existence of emergent behavior before it adversely affects system operation or safety. What should the RTA system monitor so that adverse emergent behavior can be detected is still a research question. The fleet of agents that make up the SoS may be decentralized with each agent controlling itself, yet all agents have a common goal or have knowledge of the goals of the agents each is interacting with. Therefore, each agent has its own planning modes or funnels, but they must avoid each other (in some fashion such as collision, or to not affect their nearest neighboring agent's own operations). Also, there may be environmental factors as well, such as "obstacles" or no-fly zones. By developing such models, we hope to generate adverse emergent behavior, (e.g. collision avoidance may result in *traffic jams*).

#### B.1 Characteristics of Emergent Behavior

Emergent behavior in complex systems of systems (SoS) is a topic of interest to the Air Force and this phenomenon have been widely studied [Mogul 2006], [C. Johnson 2008], [Buhr 1998], [Mataric 1993], [Boccaro 2004], [Parunak 1997].

Some characteristics of emergent behavior are listed below:

1. An emergent behavior or emergent property can appear when a number of simple entities (agents) operate in an environment, forming more complex behaviors as a collective. Why emergent behaviors occur include intricate causal relations across different scales and feedback, known as interconnectivity. The emergent property itself may be either very predictable or unpredictable and unprecedented, and represent a new level of the system's evolution. The complex behavior or properties are not a property of any single such entity, nor can they easily be predicted or deduced from behavior in the lower-level entities, and might in fact be irreducible to such behavior.
2. There is a concept of weak versus strong emergent behavior. Weak emergence is a characteristic that can be modeled and is attributable to constituent system properties (e.g. system A makes system B "go faster" by providing some energy that system B can use (w/o system A, system B "goes slower"). However, strong emergence is a characteristic that cannot be predicted or modeled and is a result of the interactions of all the systems, and is not directly traceable to the constituent properties. In other words, for linear systems the whole is equal to the sum of its parts, whereas for strong emergent behavior systems the whole is greater than sum of its parts.
3. Various forms of nonlinearity are responsible for emergent behavior. Three of the most common sources of nonlinearity are capacity limits (e.g. control saturation, taking resources from an "empty bin" or feeding resources to a full bin), feedback loops, and temporal delays. Coupling is another source of emergent behavior where, for example, one agent draws on



resources from a common bin that affects other agents (reduces their resources) – e.g. electric power for many devices.

### **B.1.1 Examples of Emergent Behavior**

Some examples of emergent behavior are listed below:

1. Game of chess: The game has a set of rules, but the future outcome cannot be predicted by the set of rules. Each player has a logical, organized, purposeful objective (i.e. not random acts). As the losing player loses pieces, his thought processes may change and emergent behavior sets in (e.g. he panics and begins to make wrong decisions). He may generally play better when he is winning.
2. Voltage supply for a collection of high but intermittent loads, such as spot welding guns. If too many guns fire at once, the supply will be unable to provide the current needed across all guns, and some of the resulting welds will be defective. In one corrective strategy, the controller for each gun monitors the line voltage, and only fires when the voltage is above a certain threshold. When this strategy is applied to all the guns in a facility, the entire set of controllers will sense a low voltage condition and delay their firing. Then, when the voltage rises, they will all recognize the availability of power and fire nearly simultaneously, pulling the voltage down and leading to another mass withdrawal. As a result of this feedback among the controllers, the system will tend to oscillate between overload and under-utilization. This is an example of coupling in the system and feedback causing emergent behavior.
3. Emergent behavior can come about in decentralized control systems. Each agent only controls its own system but interacts with other agents. There is no centralized control, only knowledge of the “common goal.” The shape and behavior of a flock of birds or school of fish are good examples of decentralized control systems. Traffic patterns in cars or internet usage are also examples of emergent behavior within their decentralized systems, resulting in traffic jams or usage bottlenecks, etc. Material handling systems can exhibit emergent behavior in the form of “traffic jams” as well.
4. Other examples include: Sand pile or avalanche model, genetic algorithms, chaos/fractals, lattice gas model, and epidemic models. Mechanical deterioration is a temporal nonlinear phenomenon that can cause system dynamics to change over time, which may result in emergent behavior.

Our review of emergent behavior led us to investigate flocking models for fleets of agents. This is discussed next.

### **B.2 Flocking Dynamic Models**

Flocking dynamics of birds or general agents have been studied extensively [Dutta 2010], [Chazelle 2012], [Dalmao 2011], [Yong 2013], [T. Johnson 2010]. Some of the fundamental characteristics of flocking are as follows:

1. Heading or directional averaging: agents tend to move in the general direction of their nearest neighboring agents.

2. Velocity averaging: agents tend to move at approximately the same speed of their nearest neighboring agents.
3. Attraction: agents tend to move closer to their nearest neighboring agents.
4. Repulsion: agents tend to move away from their nearest neighboring agents once a certain critical flock density is reached.

There have been a number of simulation models developed for flocking dynamics and we investigated some of the key aspects of the above characteristics by constructing our own flocking model.

### B.2.1 Flocking Simulation Model Development

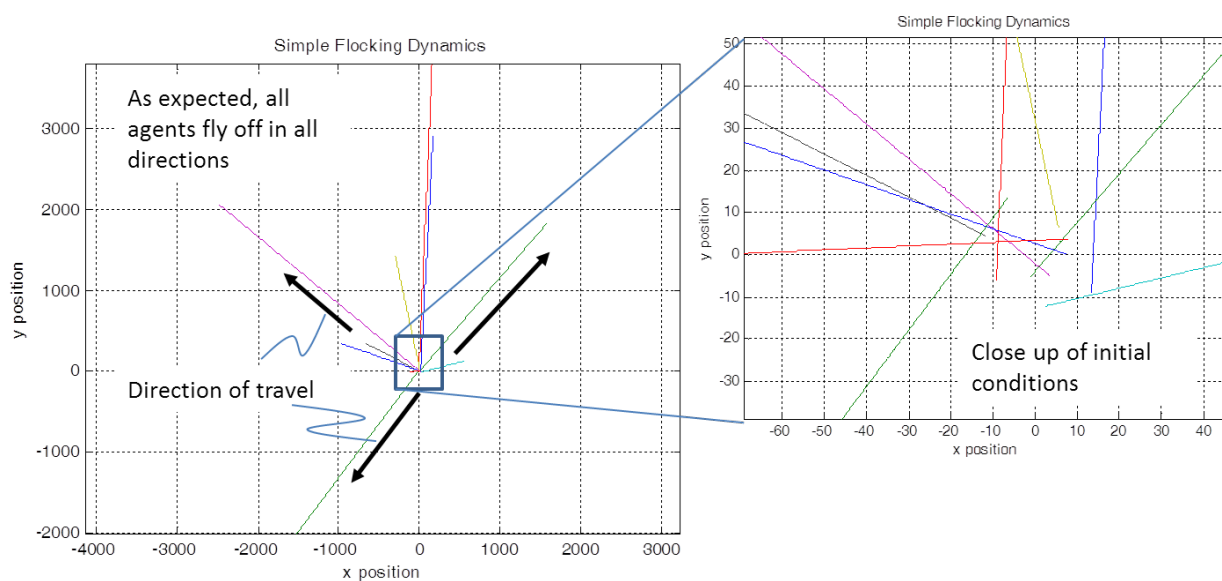
We developed a flocking model in Matlab and started with a baseline model with the following characteristics:

#### Case 1. Baseline Case:

1. Two-dimensional model: modeled motion in  $\{x,y\}$  coordinates only.
2. 10 agents starting in all different heading directions
3. No noise, no collision avoidance, no velocity limiting
4. No attraction, or heading & velocity averaging (i.e. no flocking modeled)
5. Simulated for 1000 seconds, time step = 0.1, with velocities ranging from 0.1 fps to 3.9 fps
6. Governing equations for each agent are:

$$\begin{aligned} x(t+1) &= x(t) + v(t) \cos(\theta(t)) \\ y(t+1) &= y(t) + v(t) \sin(\theta(t)) \end{aligned} \tag{B.1}$$

This gave us the results shown in Figure B.1.



**Figure B.1. Case 1: Initial Baseline Model Simulation**

To add in flocking behavior, we started with the Vicsek model [Yong 2013]:

$$\begin{aligned}x(t+1) &= x(t) + v(t) \cos(\theta(t)) \\y(t+1) &= y(t) + v(t) \sin(\theta(t)) \\ \theta(t+1) &= \theta_{avg}(t) + w_{noise}\end{aligned}\tag{B.2}$$

Here, for each agent its heading is averaged with all other agents within a specified range or region of influence, which we set as  $R_{inf} = 10$  ft. To this model we also included velocity averaging, and had the averages “slowly” affect motion by including relative weightings on the previous values for heading and velocity and their averages. This gives,

$$\begin{aligned}x(t+1) &= x(t) + v(t) \cos(\theta(t)) \\y(t+1) &= y(t) + v(t) \sin(\theta(t)) \\ \theta(t+1) &= (1 - w_{\theta})\theta(t) + w_{\theta}\theta_{avg}(t) + w_{\theta\_noise}, \quad 0 \leq w_{\theta} \leq 1 \\ v(t+1) &= (1 - w_v)v(t) + w_v v_{avg}(t) + w_{v\_noise}, \quad 0 \leq w_v \leq 1\end{aligned}\tag{B.3}$$

#### Case 2. Velocity and Heading Averaging:

1. Same scenario as Baseline – 10 agents starting in same directions as baseline
2. No noise, no collision avoidance, no velocity limiting
3. No attraction, but 10% heading and velocity averaging ( $w_{\theta} = w_v = 0.1$ )

This gives us the results shown in Figure B.2. Here it can be seen that due to the defined region of attraction, the flock breaks off into two subgroups with different average headings. Also, even with only a 10% weighting on the averaging influence, the agents within each subgroup very quickly converge to nearly the same velocities and headings. The velocities for subgroup #1 and #2 converged to ~1.75 fps and ~2 fps, respectively.

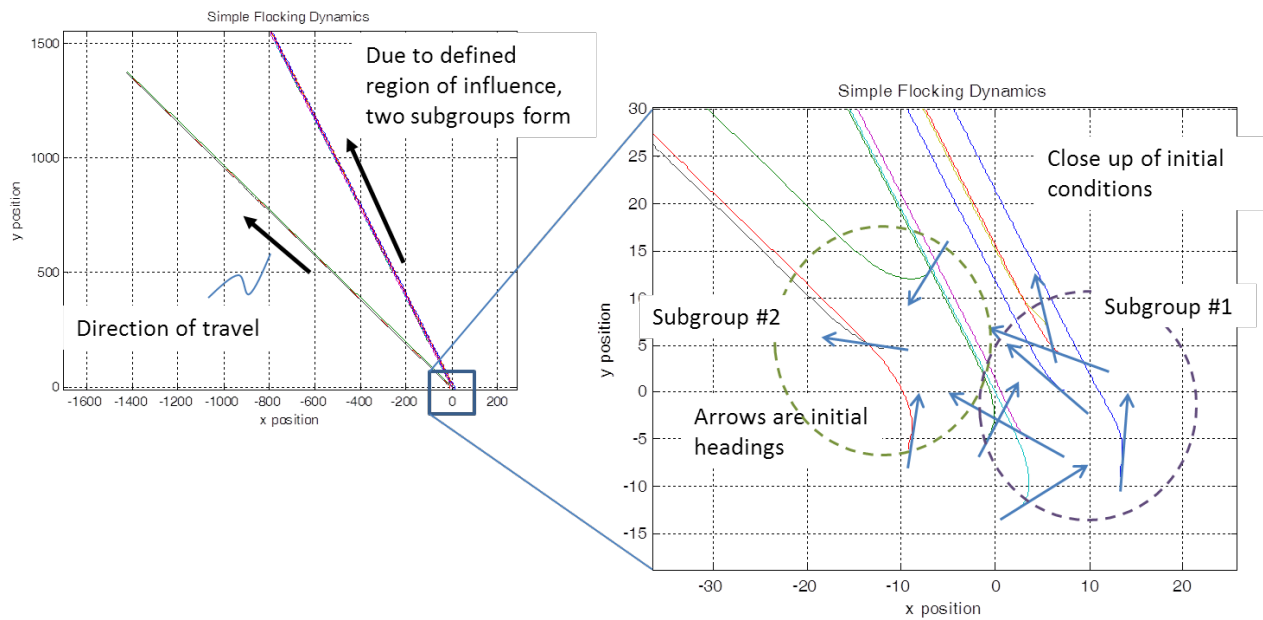
#### Case 3. Noise Added:

1. Same scenario as Case #2 – 10 agents, 10% heading & velocity averaging
2. No collision avoidance, no velocity limiting, no attraction
3. Now add in noise (at “reasonable levels”)
4. Random numbers for noise models “seeded” for repeatable results

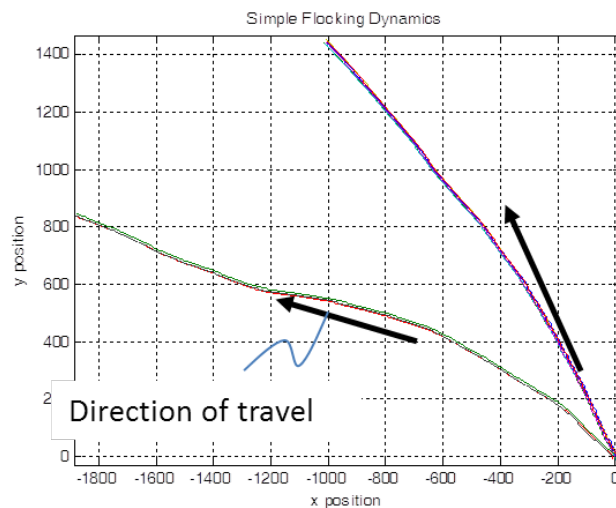
This gives us the results shown in Figure B.3. Here, it can be seen that the noise now gives more realistic motion and the subgroup paths are no longer straight lines. The heading and velocity time histories for the two subgroups are shown in Figure B.4.

We then investigated limiting the velocity, where noise on velocity was always  $> 0$  until it reached a predefined maximum value,  $V_{max}$ , then the noise on velocity was always  $< 0$  until it reached a predefined minimum value,  $V_{min}$ , then the process was repeated. This gave a “sawtooth” profile to the fleet’s average velocities. However, this had little effect on the resulting motion of the fleet. As long as all agents have approximately same average velocity, then they will act as a flock. Therefore, at this point we no longer considered special velocity profiles and simply set a

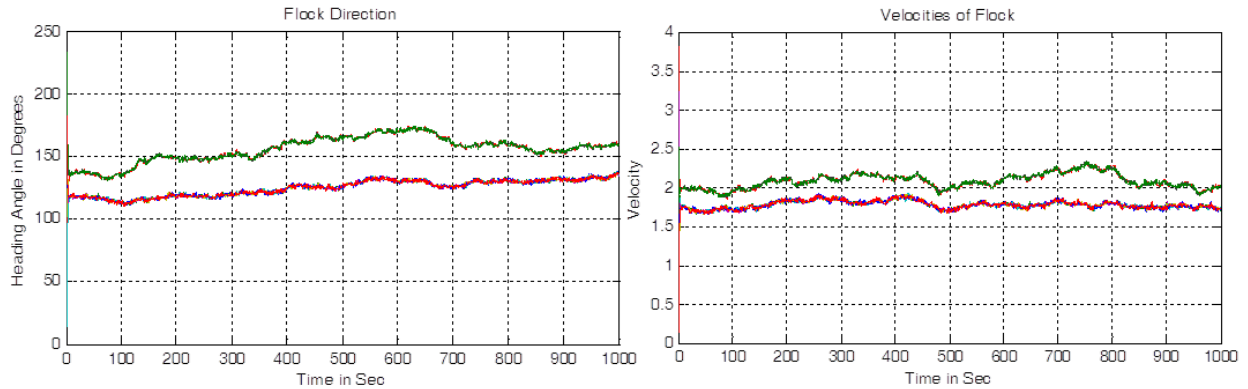
limit of  $\pm 10$  fps. Due to the random noise inputs and averaging, the velocities were never observed to go unbounded and never reached the defined limit.



**Figure B.2. Case 2: Velocity and Heading Averaging**



**Figure B.3. Case 3: Noise Added to Governing Equations, No Attraction**

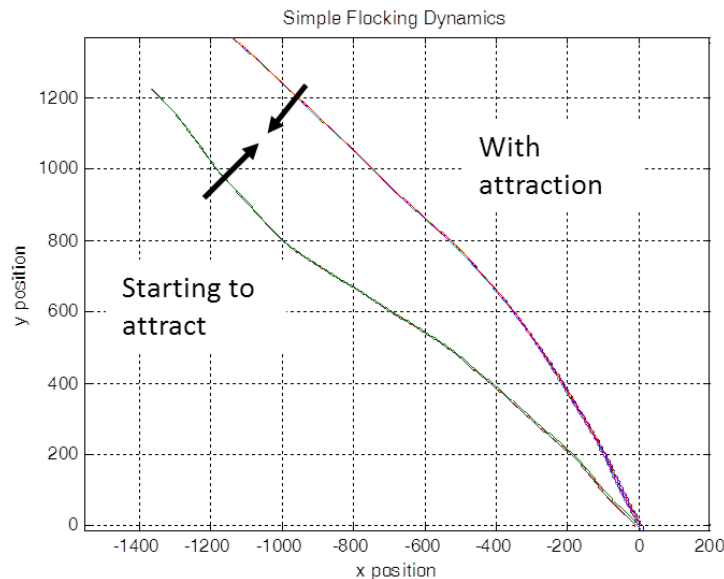


**Figure B.4. Heading and Velocity Time Histories of Each Subgroup for Case 3**

Case 4. Weak Attraction:

1. Same scenario as Case #3 – 10 agents, 10% heading & velocity averaging
2. No collision avoidance,  $\pm 10$  fps velocity limiting
3. Noise added at same seed values
4. Attraction added at a value of  $w_a = 0.002$

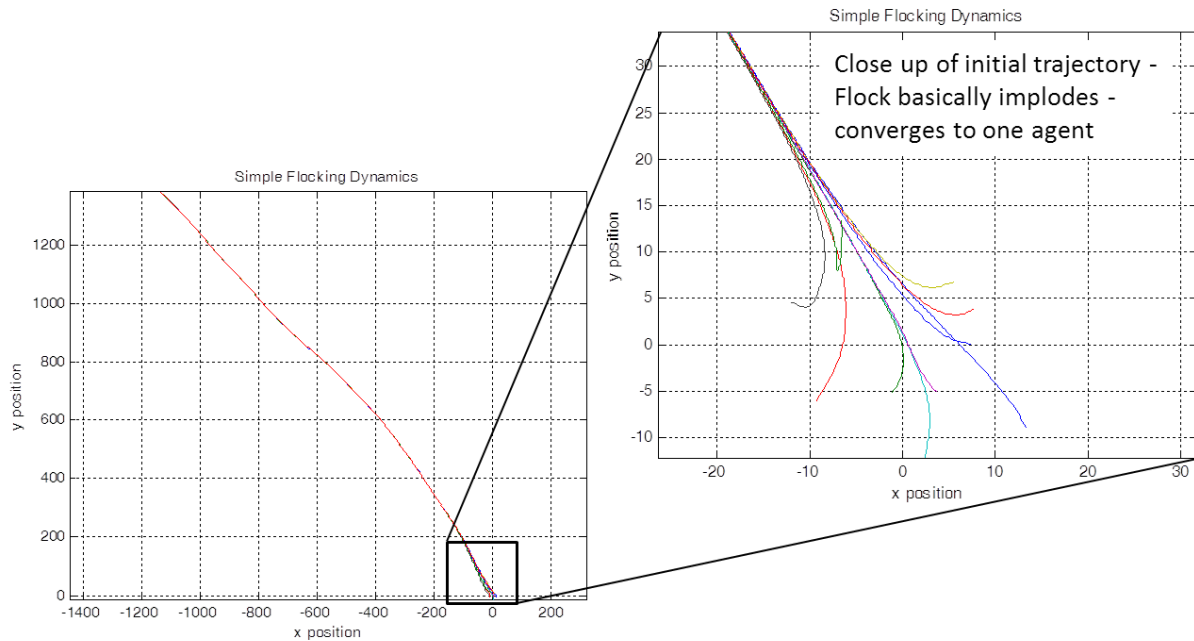
Here, we used the Boids pseudo-code model [boids 2002] in which each agent has a perceived flock center of mass (c.m.) and each agent flies toward that perceived c.m. with some weighting,  $w_a$ . This case gives the results shown in Figure B.5. Comparing this result with that shown in Figure B.3, it can be seen that the two subgroups are slowly starting to attract to each other. If the simulation were to continue, these two subgroups would eventually merge into one group.



**Figure B.5. Case 4: Weak Attraction Added**

### Case 5. Strong Attraction:

Increasing the attraction weighting by 100 times, ( $w_a = 0.2$ ) gives the results shown in Figure B.6. Here, it can be seen that this level of attraction causes a “black hole” effect in which all agents quickly converge to one point mass, with no chance to form separate subgroups. This level of attraction will not yield interesting flocking behavior.



**Figure B.6. Case 5: Strong Attraction Added**

### Case 6. Mid-Range Attraction:

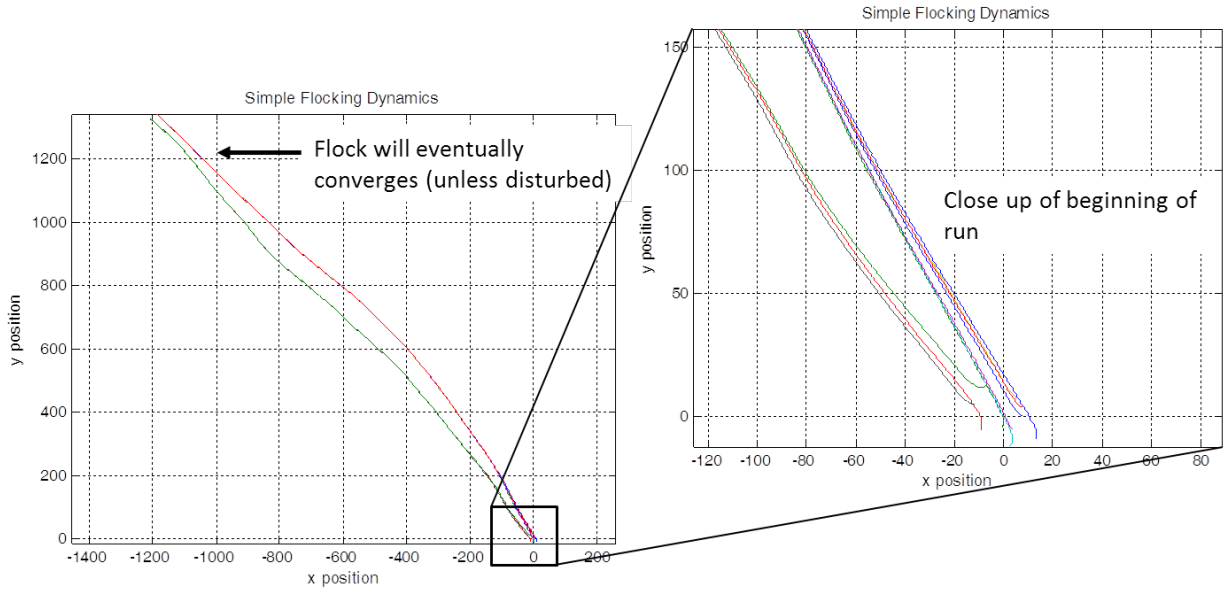
We found that an attraction weighting of  $w_a = 0.01$  gives results in which two subgroups are still formed initially, but the attraction is strong enough to be evident within the simulation time. This case is shown in Figure B.7.

### Case 7. Add in Collision Avoidance:

With nonzero attraction, all agents will eventually converge to a single point mass. Given that these agents will eventually be defined as physical objects (i.e. UASs), this model is clearly unrealistic. Therefore, we added a simple collision avoidance routine to keep each agent a reasonable distance from each other. We began by defining a collision avoidance radius in which each agent was to keep that distance from all neighboring agents. This distance was set as  $R_{ca} = 1$  ft, of course defined to be smaller than the region of influence (recall,  $R_{inf} = 10$  ft).

We first considered a “weak” collision avoidance model in which the collision avoidance radius could be violated, but the agents tended to stay away from each other by turning their headings from each other by a pre-defined amount. However, this led to a heading angle limit cycle. Once outside the collision avoidance radius, each agent would then turn their headings back toward each other due to the heading averaging. Therefore, this collision avoidance scheme was

not pursued. Instead, we constructed a “strong” collision avoidance model in which the collision avoidance radius was not allowed to be violated (in the limit). Here, if one agent is a distance  $d$  from another, in which  $d < R_{ca}$ , then from simple geometrical relations, both agents are “nudged” an equa-distance from each other to where  $d = R_{ca}$ . This is shown in Figure B.8.



**Figurer B.7. Case 6: Mid-Range Value for Attraction Added**

For an agent  $i$  with position  $\{x_i, y_i\}$  and an agent  $j$  with position  $\{x_j, y_j\}$ , it can be shown that the value of the “nudge”  $n$  is the smallest root of the 2<sup>nd</sup> order polynomial:

$$0 = 2N^2 + 2(x_i - x_j + y_i - y_j)N + x_i^2 + y_i^2 - 2(x_i x_j + y_i y_j) + x_j^2 + y_j^2 - R_{ca}^2 \quad (\text{B.4})$$

Therefore, each agent is moved by  $\frac{1}{2}n$  in the opposite direction from the other. Although it can be argued that this approach is physically unrealistic due to the instantaneous change in positions for the agents, we found in simulation that this was not evident as the “nudges” always tended to be small. Note that as two agents are nudged away from each other there is the possibility that they can be moved within the collision avoidance radius of other neighboring agents. Therefore, this nudging process should be looped through the entire fleet until no collision avoidance radius is violated between any two agents. However, this would be computationally time consuming, so we limited the process to two or three passes through the fleet. In this case, there is the possibility that the collision avoidance radius could potentially be violated (e.g. nudging agent  $j$  away from agent  $i$  may move agent  $j$  to within  $R_{ca}$  from agent  $k$ . If indice  $k < j$ , then that will be missed without another pass through the fleet). However, again, we found that in the simulation results there was no evidence of this occurring, and in steady state all agents tended to a distance of  $R_{ca}$  from each other, and this is shown in Figure B.9 and Figure B.10.

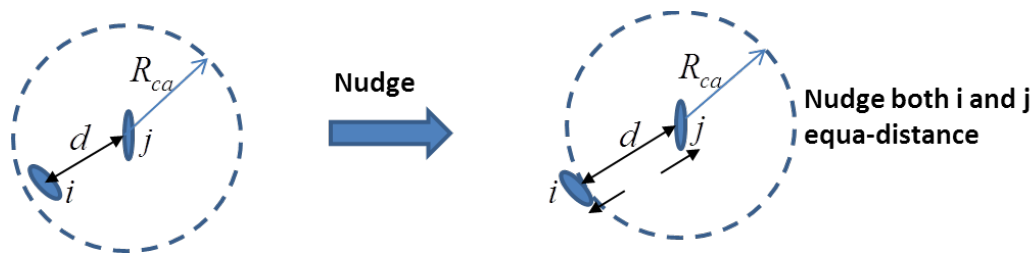


Figure B.8. “Strong” Collision Avoidance Scheme

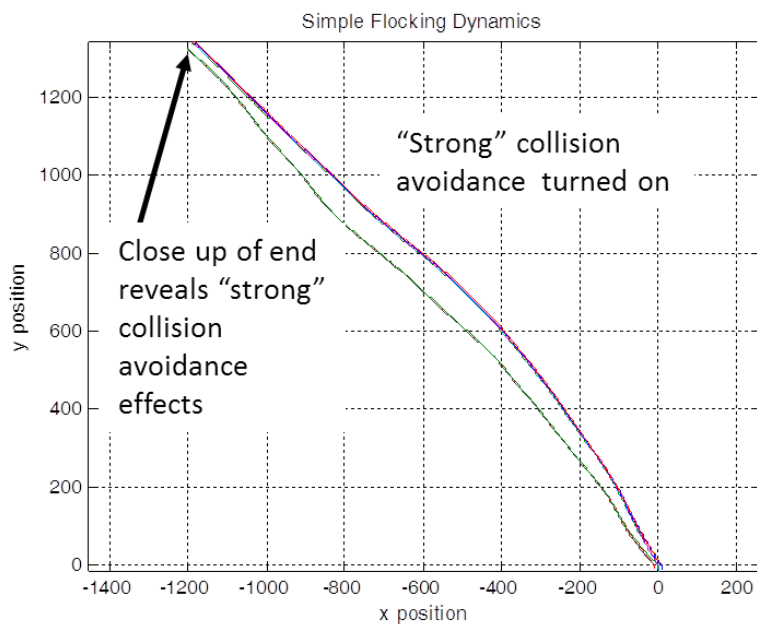


Figure B.9. Case 7: “Strong” Collision Avoidance Added

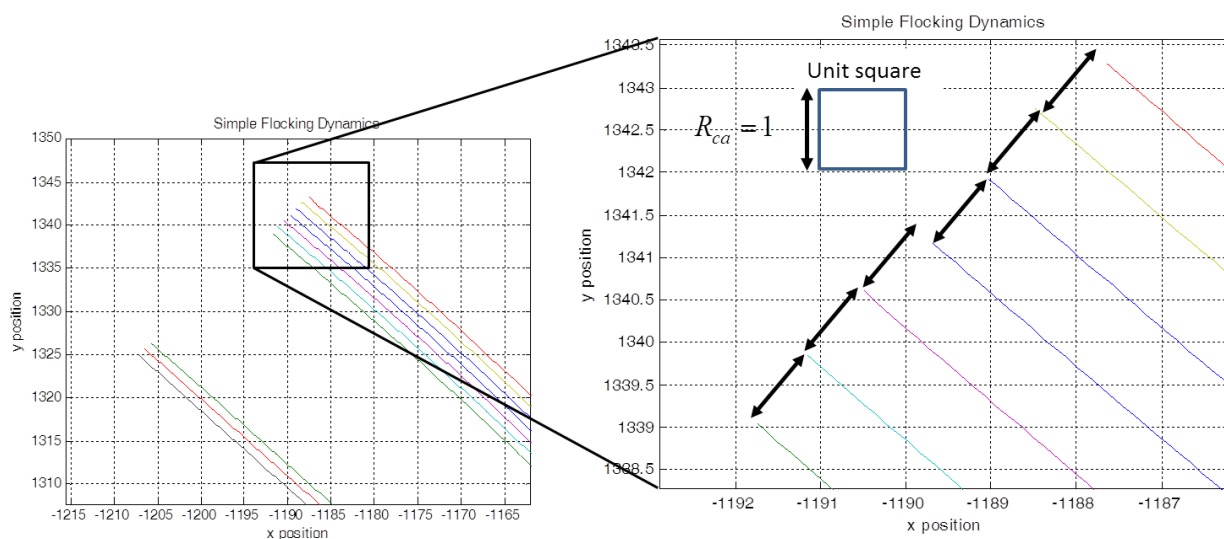
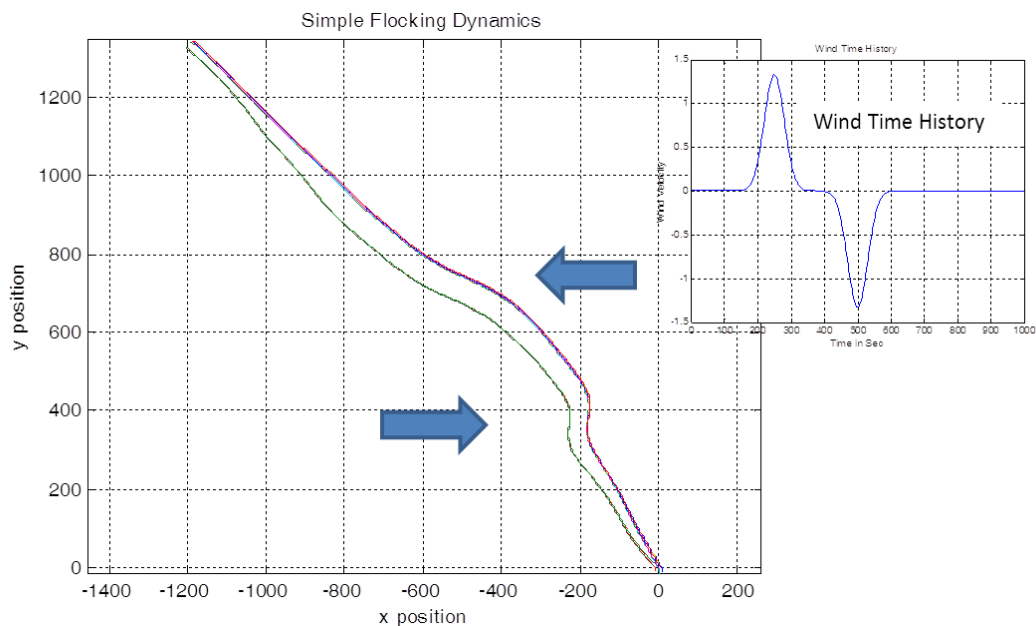


Figure B.10. Case 7: “Strong” Collision Avoidance – Close Up of Steady State

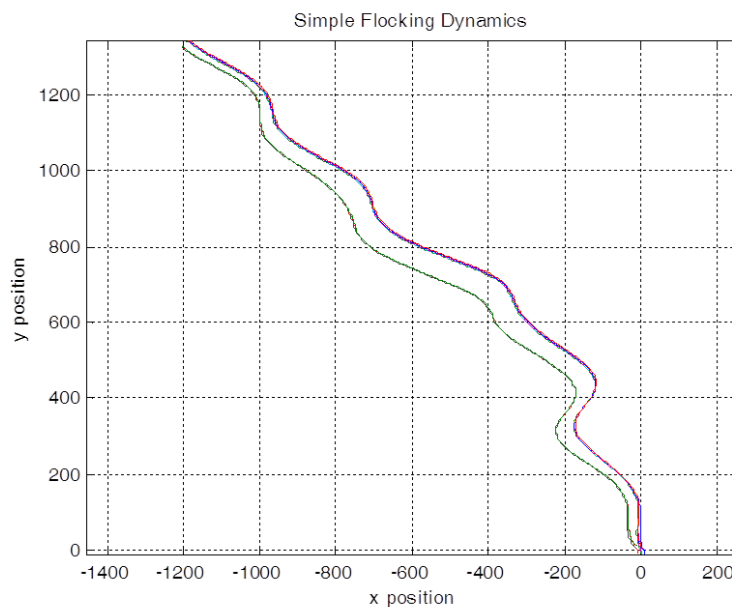


### Case 8. Adding Wind and Other Disturbances:

Starting with the model in Case 7, we next added wind disturbances to the fleet in the x-direction. This is shown in Figure B.11. We then added random sinusoidal (sum of sine waves) disturbances, and this is shown in Figure B.12.



**Figure B.11. Case 8: Winds in X-Direction Added**

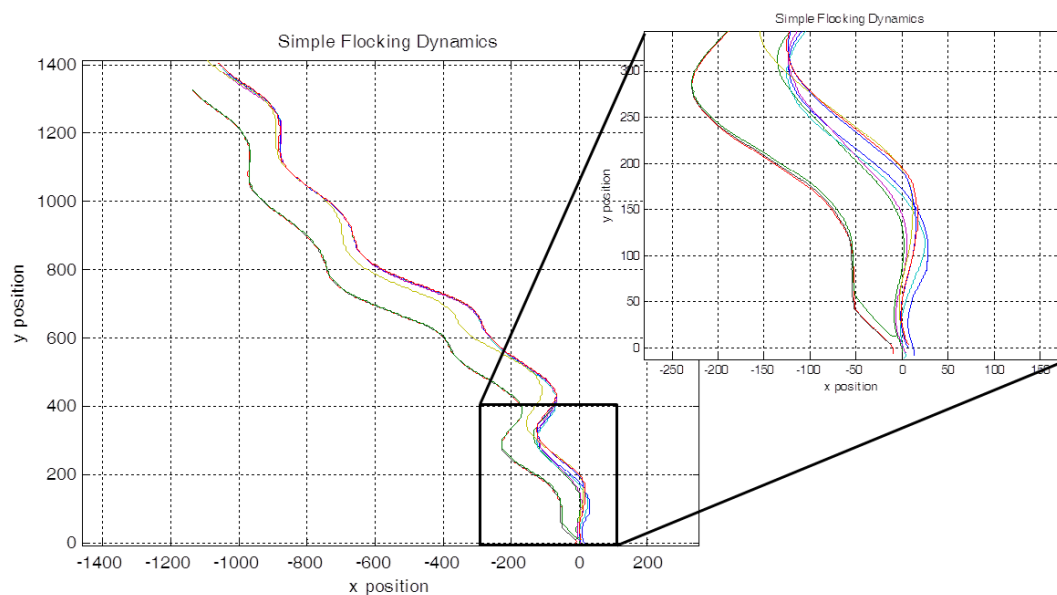


**Figure B.12. Case 8: Adding Random Sinusoidal Disturbances**

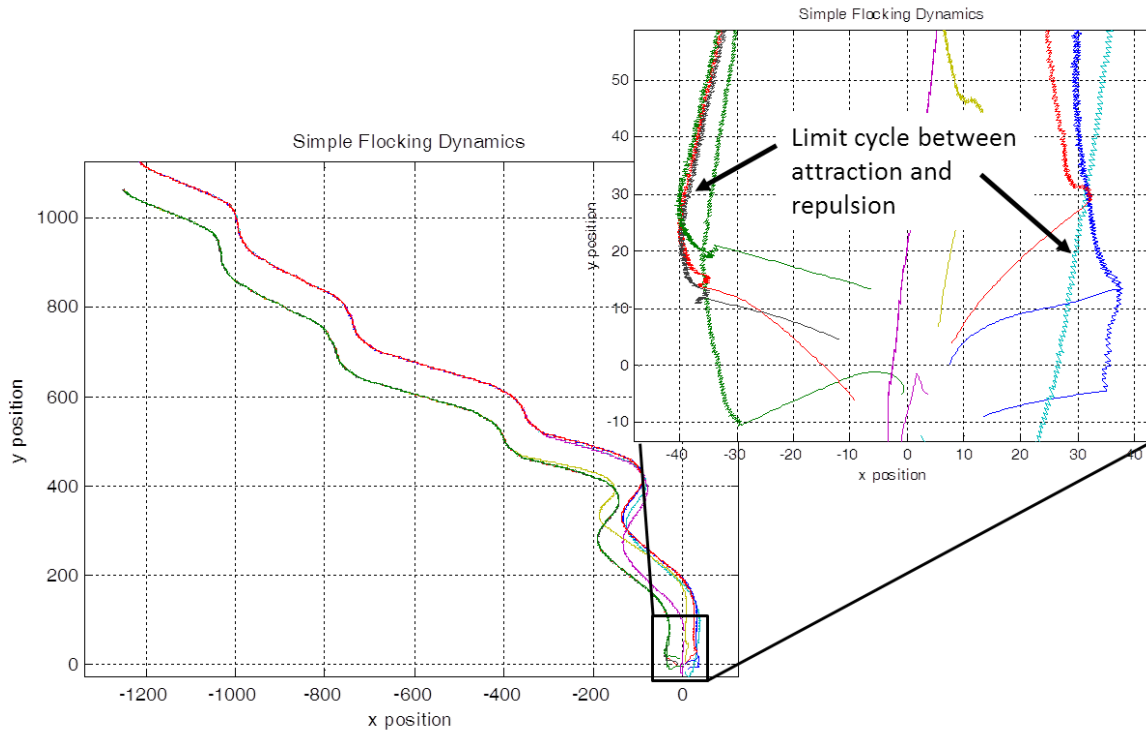
### Case 9. Repulsion Added:

It is noted that with the characteristics of attraction and heading and velocity averaging, the flocking trajectories show a comparatively fast tendency toward steady state in which all agents are simply flying next to each other at a distance  $R_{ca}$  from each other. This does not lend itself toward more interesting studies that may include emergent behavior.

Therefore, in an attempt to obtain more interesting results we added model repulsion characteristics to the flock. Here, if the distance of an agent from the flock's perceived center of mass is less than a predefined distance,  $R_{rep}$ , then we change the sign on the "attraction contribution" to the total velocity. That is, we use the same model as for attraction, but simply change the sign when the above condition is met. Then, when the agent's distance from the flock's perceived center of mass is greater than  $R_{rep}$ , attraction is turned back on. Figure B.13 shows an initial result when repulsion is added to the simulation. It can be seen that this initially causes more "activity" with the agents' trajectories. However, if the gain on the repulsion characteristic is increased by 10, then a strong limit cycle between attraction and repulsion results. This is shown in Figure B.14. One approach may be to add a hysteresis factor so that the "switch" between attraction and repulsion is not instantaneous. Although we did not investigate this, we did vary the relative gains between attraction and repulsion and these results are shown in Figure B.15. Here, we show results for equal weightings on attraction and repulsion as well as cases for when attraction is twice as strong as repulsion and vice-versa, when repulsion is twice as strong as attraction. As expected, when repulsion is twice as strong as attraction the flock tends to have more "activity" and does not reach steady state within the simulation time. However, if the simulation is run for a longer period of time, then the same steady state characteristics of all agents flying next to each other with the same velocities and same headings does eventually result.



**Figure B.13. Case 9: Repulsion Added**



**Figure B.14. Repulsion Gain Increased**

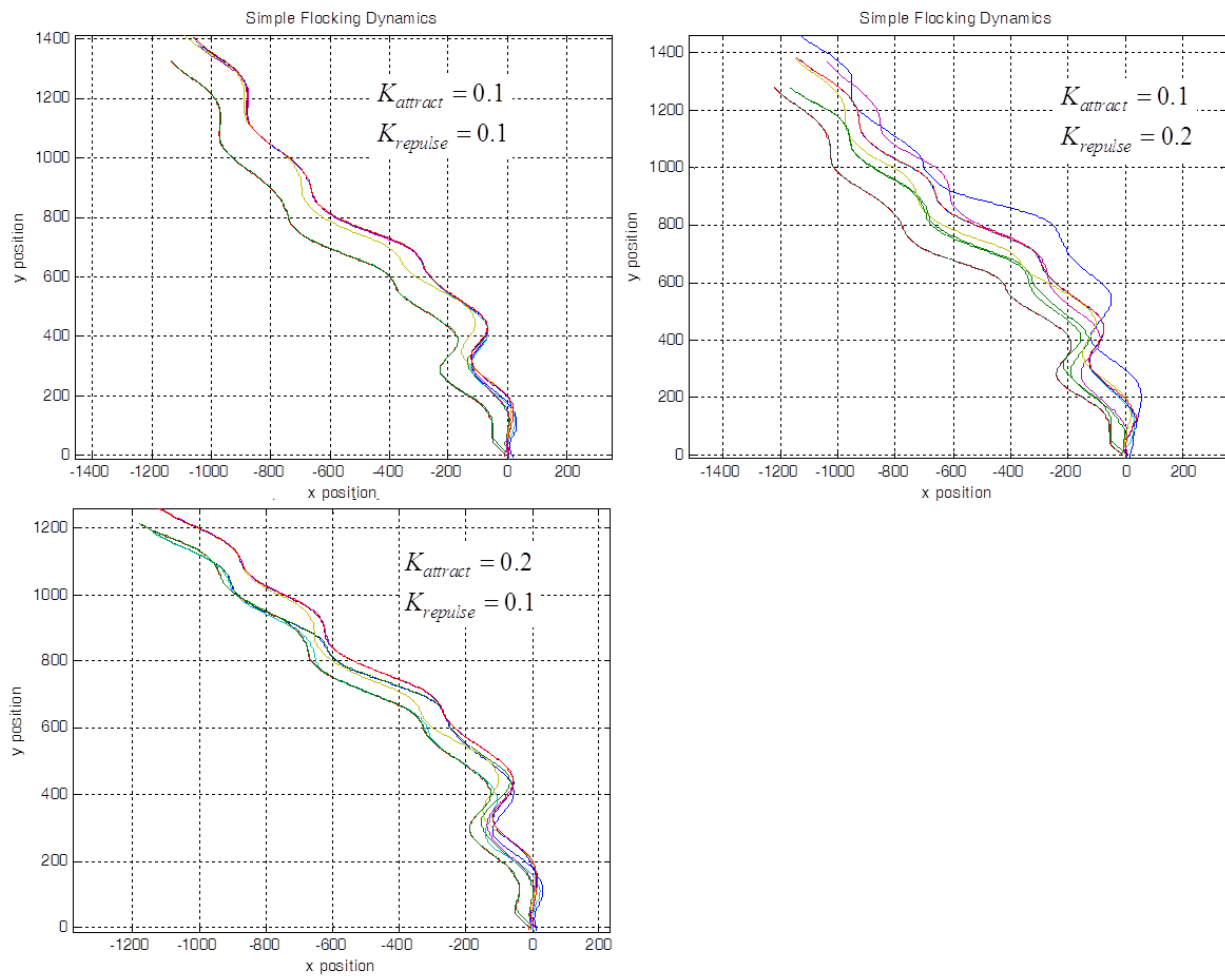
#### Case 10. Heterogeneity Added:

In an attempt to further add more interesting characteristics to the flocking dynamics, we next added heterogeneity to the fleet. That is, each agent has different gains on attraction, repulsion and heading & velocity averaging. This gives some agents “slow” flocking dynamics and others “fast” dynamics. For random variations on these values for each agent, one sample result is shown in Figure B.16. Here, clearly steady state has not been reached within the simulation run time, and in fact, it looks as though the flock is diverging. The heading and velocity time histories of the agents are shown in Figure B.17 which also show there is no strong averaging effects.

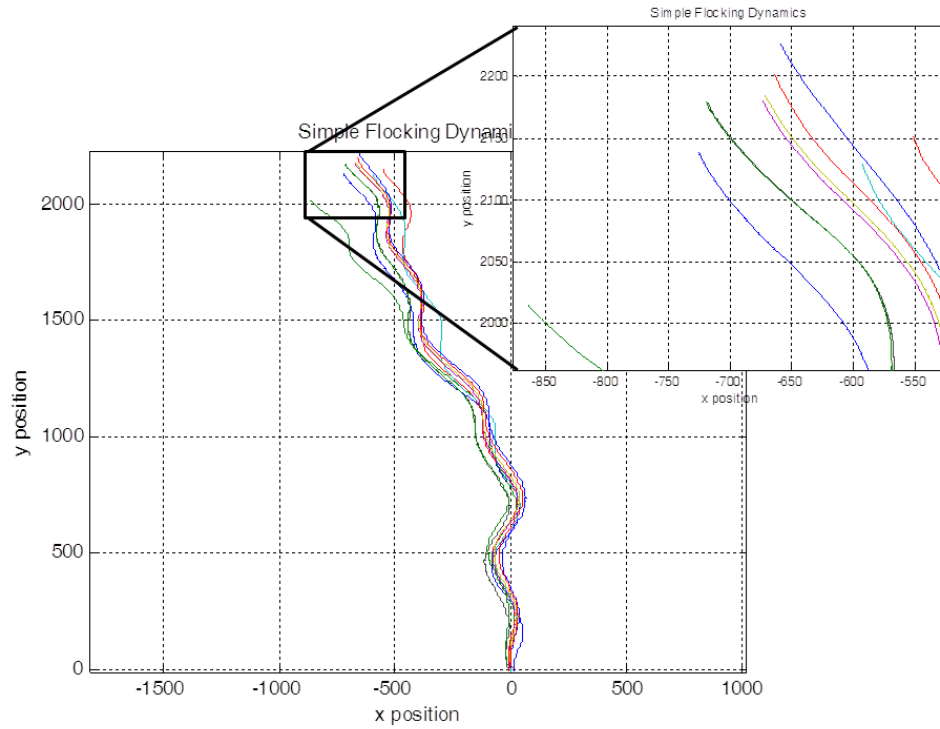
However, note that too much variation in the agents’ characteristics will result in less flocking behavior and more random motion. This is analogous to having different species of birds fly together in a flocking motion. It is not an event observed in nature. Therefore, there needs to be an “optimum” set of model characteristics that will mimic flocking seen in birds.

#### Case 11. Controlling the General Flock Direction:

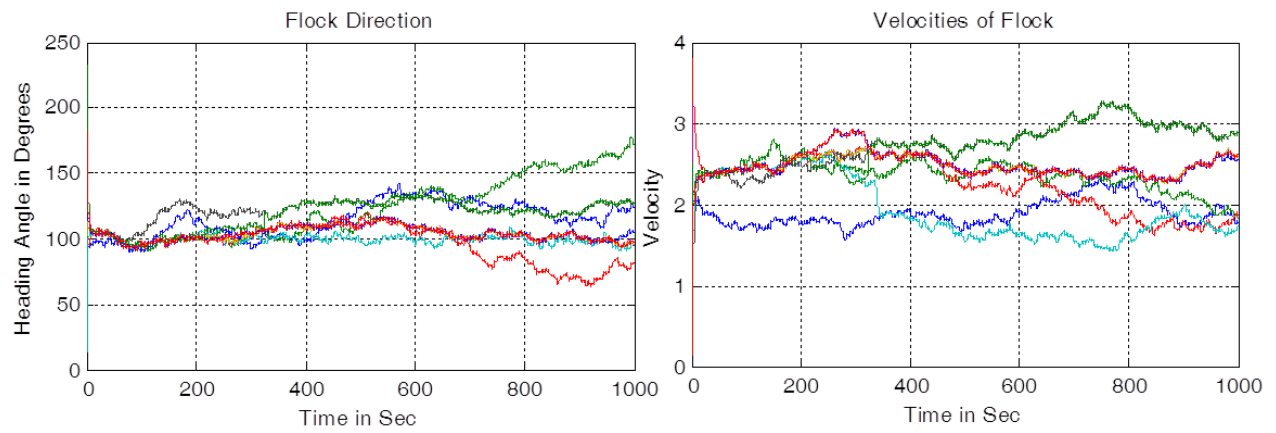
To control the general flock direction, we added a commanded velocity component in the same manner as the wind disturbances shown in Figure B.11 and Figure B.12. An example simple steady “push to the right” is shown in Figure B.18. More complex command histories can be developed, but we ended our studies of flocking dynamics with this case.



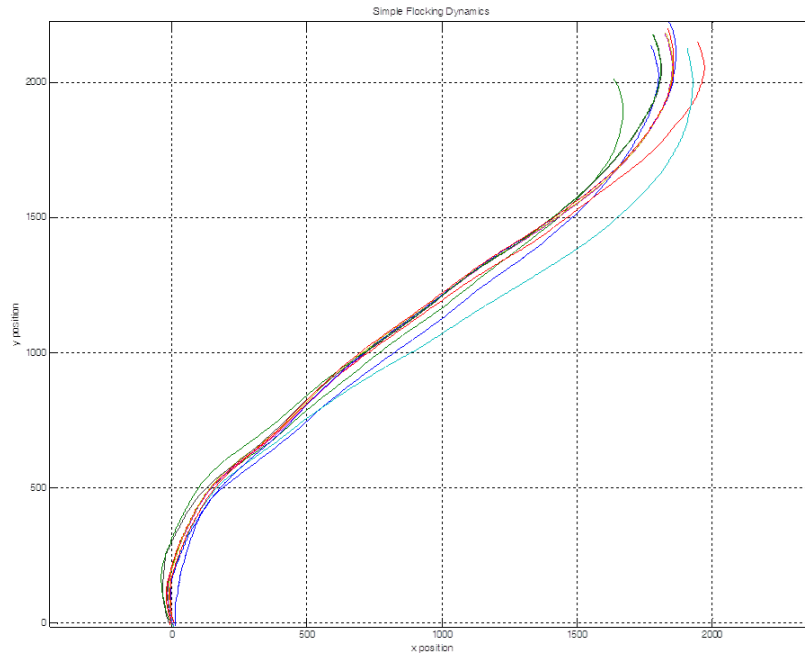
**Figure B.15. Varying Relative Weightings on Attraction and Repulsion**



**Figure B.16. Case 10: Heterogeneity Added to Flock**



**Figure B.17. Case 10: Heading and Velocity Time Histories for Heterogeneous Flock**



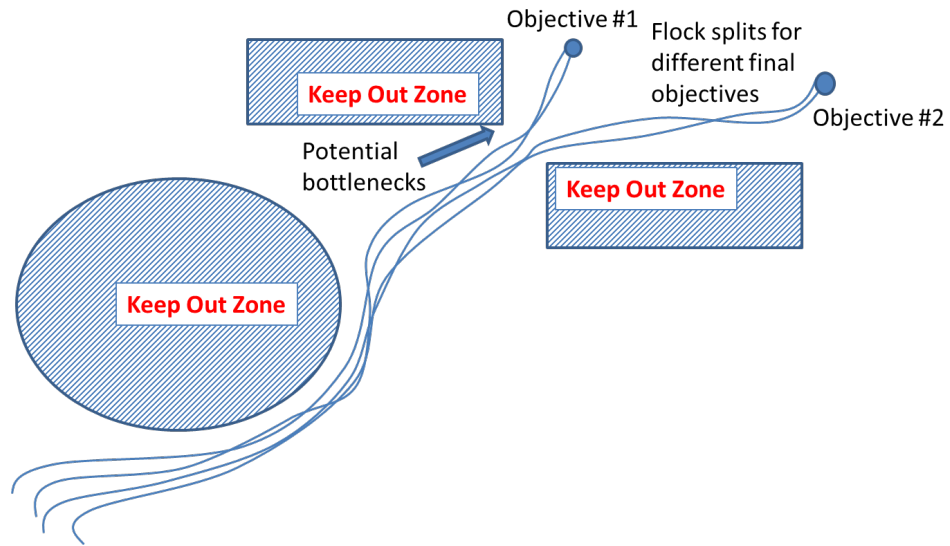
**Figure B.18. Case 11: Controlling General Flock Direction – Push to the Right**

### **B.2.2 Flocking Dynamics – Summary and Observations**

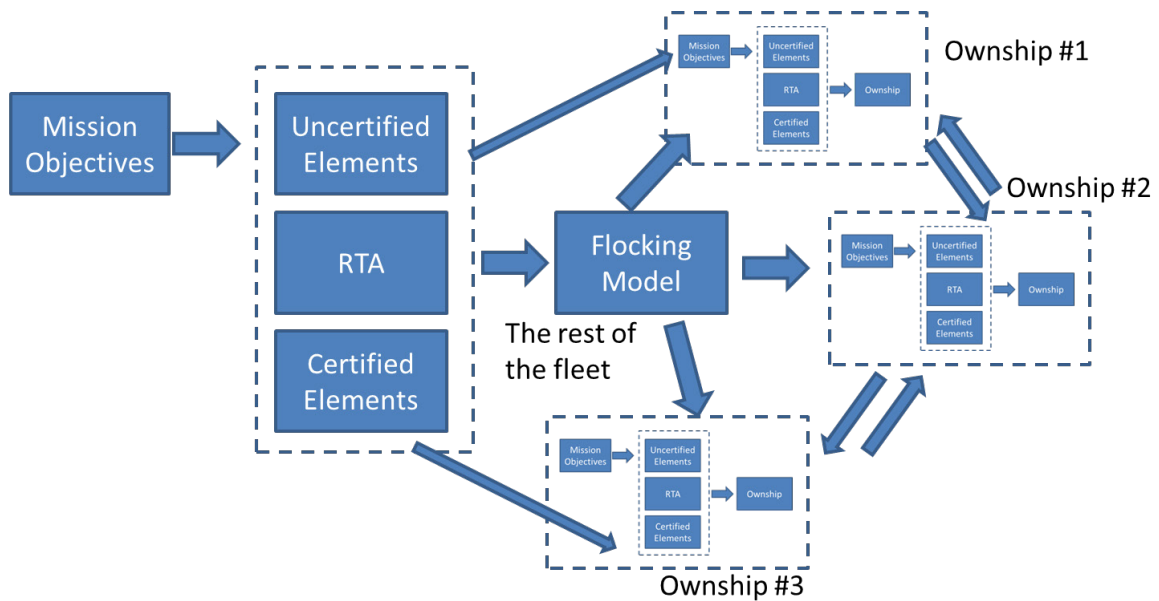
Our studies on flocking dynamics show that without some external disturbance the flocking trajectories quickly tend toward a more deterministic steady state flight. However, to eventually study fleets of UASs in some flocking dynamic mission, we will want to introduce commanded trajectories or waypoint objectives, and this may further cause randomness characteristics of the flocking behavior to be quickly minimized. Figure B.19 presents an illustration of a flocking mission of UASs, which may be commanded to avoid keep out zones and fly to certain objective locations. Fundamentally, all agents start to do what they are commanded to do. The key question then becomes: how do we introduce mission objectives, yet still keep some randomness/flocking behavior?

Figure B.20 presents an initial mixed centralized/decentralized command and control framework for a flocking UAS fleet. Here, there may be some centralized command structure that delivers overall fleet mission goals that interacts with the flocking dynamic model, which in turn interacts with each individual ownership. Each ownership itself will have its own individual command/control elements. Both at the centralized and decentralized levels, there may exist certified and uncertified elements, requiring RTA systems for operational protection from software errors.

Once the flocking dynamic studies were completed, the decision was made to simply investigate a fleet of UASs in some type of coordinated control mission. The focus is more on actual UAS vehicles performing more realistic flight characteristics, rather than mimicking the flocking motions of birds in nature.



**Figure B.19. Illustration of Flocking Mission**



**Figure B.20. Proposed Command and Control Framework for Flocking UAS Fleet**

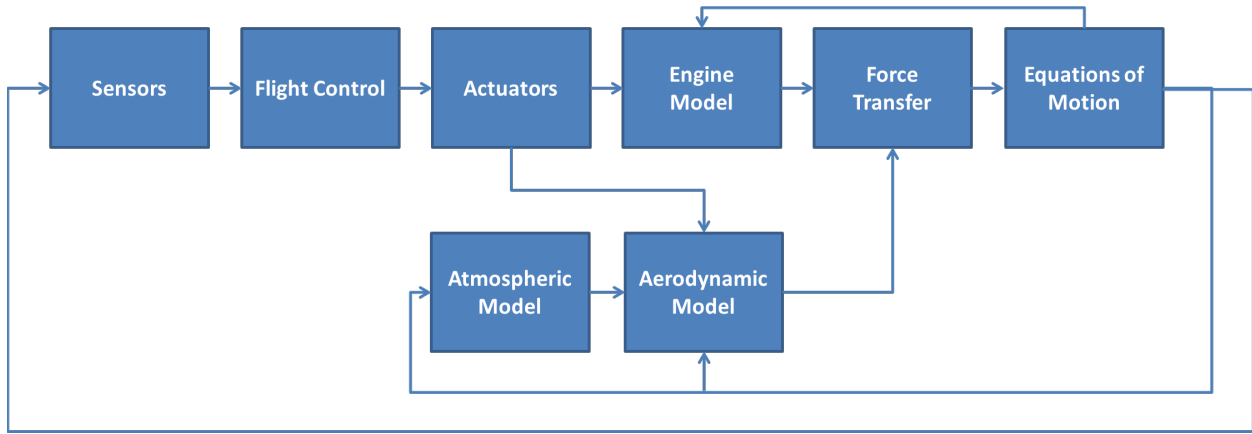
## Appendix C

### Morphing Wing Model

In this appendix, we present the morphing wing 6-DOF dynamic model and transition controller design.

#### C.1 Equations of Motion

Figure C.1 below shows the major components of the simulation used for the inner-loop RTA challenge problem.



**Figure C.1. Full Simulation Architecture used for Inner-Loop RTA Challenge Problem**

Figure C.2 shows the simplifications that are made to the simulation architecture to enable reachability analysis. The 'Sensors' and 'Actuators' block is reduced to a pass-through (that is, signals from the 'Equations of Motion' are passed directly into 'Flight Control' and commands from 'Flight Control' are passed directly into the 'Aerodynamic Model' and 'Force Transfer'). The 'Atmospheric Model' is simplified to output a constant air density (corresponding to the air density at 1500 feet). The 'Aerodynamic Model' is simplified by reducing the dimensionality of the model and using polynomial fits for the data instead of a look-up table.

The following subsections provide greater detail for each of the components in the simplified simulation architecture.

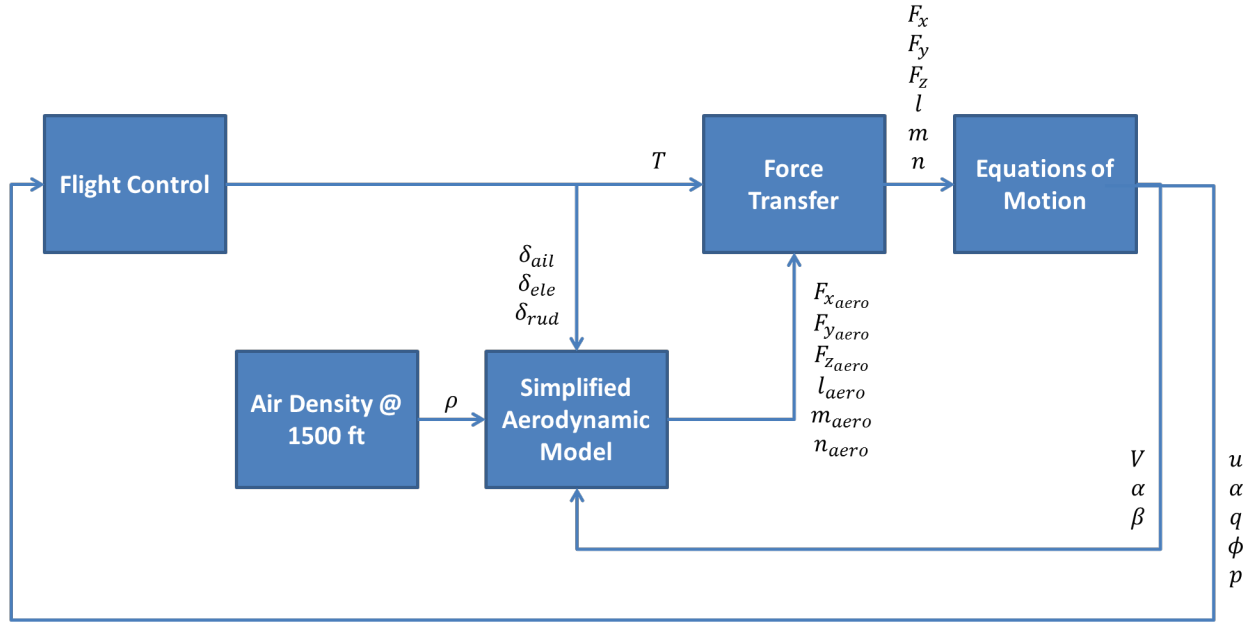
The six degree-of-freedom rigid-body flat-earth fixed-wing aircraft equations of motion are given below [Stevens 1992]. The states are:  $[u \ v \ w \ p \ q \ r \ \phi \ \theta \ \psi \ N \ E \ h]$ .  $c_1$  through  $c_9$  used in Equations (C.4)-(C.6) are derived from the moments of inertia and are calculated in Equations (C.13)-(C.22).

$$u = rv - qw - g \sin \theta + \frac{F_x}{M} \quad (C.1)$$

$$\dot{v} = -ru + pw + g \sin \phi \cos \theta + \frac{F_y}{M} \quad (C.2)$$



$$\dot{w} = qu - pv + g \cos \phi \cos \theta + \frac{F_z}{M} \quad (C.3)$$



**Figure C.2. Simplified Simulation Architecture used to Enable Inner-Loop RTA Analysis**

$$\dot{p} = (c_1 r + c_2 p)q + c_3 l + c_4 n \quad (C.4)$$

$$\dot{q} = c_5 pr - c_6(p^2 - r^2) + c_7 m \quad (C.5)$$

$$\dot{r} = (c_8 p - c_2 r)q + c_4 l + c_9 n \quad (C.6)$$

$$\dot{\phi} = p + \tan \theta (q \sin \phi + r \cos \phi) \quad (C.7)$$

$$\dot{\theta} = q \cos \phi - r \sin \phi \quad (C.8)$$

$$\dot{\psi} = \frac{q \sin \phi + r \cos \phi}{\cos \theta} \quad (C.9)$$

$$\dot{N} = u \cos \theta \cos \psi + v(-\cos \phi \sin \psi + \sin \phi \sin \theta \cos \psi) + w(\sin \phi \sin \psi + \cos \phi \sin \theta \cos \psi) \quad (C.10)$$

$$\dot{E} = u \cos \theta \sin \psi + v(\cos \phi \cos \psi + \sin \phi \sin \theta \sin \psi) + w(-\sin \phi \cos \psi + \cos \phi \sin \theta \sin \psi) \quad (C.11)$$

$$\dot{h} = u \sin \theta - v \sin \phi \cos \theta - w \cos \phi \cos \theta \quad (C.12)$$

$$c_1 = \frac{(I_{yy} - I_{zz})I_{zz} - I_{xz}^2}{I_{xx}I_{zz} - I_{xz}^2} \quad (C.13)$$

$$c_2 = \frac{(I_{xx} - I_{yy} + I_{zz})I_{xz}}{I_{xx}I_{zz} - I_{xz}^2} \quad (C.14)$$

$$c_3 = \frac{I_{zz}}{I_{xx} I_{zz} - I_{xz}^2} \quad (C.15)$$

$$c_3 = \frac{I_{zz}}{I_{xx} I_{zz} - I_{xz}^2} \quad (C.16)$$

$$c_4 = \frac{I_{xz}}{I_{xx} I_{zz} - I_{xz}^2} \quad (C.17)$$

$$c_5 = \frac{I_{zz} - I_{xx}}{I_{yy}} \quad (C.18)$$

$$c_6 = \frac{I_{xz}}{I_{yy}} \quad (C.19)$$

$$c_7 = \frac{1}{I_{yy}} \quad (C.20)$$

$$c_8 = \frac{I_{xx}(I_{xx} - I_{yy}) + I_{xz}^2}{I_{xx} I_{zz} - I_{xz}^2} \quad (C.21)$$

$$c_9 = \frac{I_{xx}}{I_{xx} I_{zz} - I_{xz}^2} \quad (C.22)$$

The calculated quantities are derived from states in the equations of motion. Air density is calculated using an atmospheric model and will be assumed to be a constant value. Winds are assumed to be zero in this calculation.

$$V = (u^2 + v^2 + w^2)^{\frac{1}{2}} \quad (C.23)$$

$$\bar{q} = \frac{1}{2} \rho V^2 \quad (C.24)$$

$$\alpha = \arctan\left(\frac{w}{u}\right) \quad (C.25)$$

$$\beta = \arcsin\left(\frac{v}{V}\right) \quad (C.26)$$

An initial condition used to initialize the equations of motion to a straight-and-level flight condition (constant altitude, constant heading flight) is provided below.

$$u = 187.361 \text{ ft/s}, \quad v = 0, \quad w = 8.469 \text{ ft/s}$$

$$p = 0, \quad q = 0, \quad r = 0$$

$$\phi = 0, \quad \theta = 0.0452 \text{ rad}, \quad \psi = 0$$

$$N = 0, \quad E = 0, \quad h = 1500 \text{ ft}$$

## C.2 Aerodynamic Model

The aerodynamic forces and moments can be built up using the following equations [Stevens 1992]. Polynomial approximations for the aerodynamic coefficients can be found in Table C.1. In the approximations,  $\alpha$ ,  $\delta_{ail}$ ,  $\delta_{ele}$ , and  $\delta_{rud}$  are in degrees. Limits for

$\alpha$ ,  $\delta_{ail}$ ,  $\delta_{ele}$ , and  $\delta_{rud}$  can be found in Table C.2. The control derivatives  $C_{n\,dail}$ ,  $C_{L\,dele}$ , and  $C_{l\,drud}$  do not vary significantly with angle of attack. The control derivatives  $C_{l\,dail}$ ,  $C_{D\,dele}$ , and  $C_{m\,dele}$  do depend on angle of attack. To simplify the fit, polynomial approximations for all of the control derivatives were created using an angle of attack near the initial condition of  $\alpha = 3$  degrees. The aerodynamic model provided in this section assumes that sideslip is approximately zero ( $\beta \approx 0$ ).

$$C_L = C_{L_0}(\alpha) + C_{L\,dele}(\alpha, \delta_{ele}) \quad (C.27)$$

$$C_D = C_{D_0}(\alpha) + C_{D\,dele}(\alpha, \delta_{ele}) + C_{D\,drud}(\alpha, \delta_{rud}) \quad (C.28)$$

$$C_Y = C_{Y_0}(\alpha) + C_{Y\,dail}(\alpha, \delta_{ail}) + C_{Y\,drud}(\alpha, \delta_{rud}) + C_{Y\,damp} \quad (C.29)$$

$$C_l = C_{l_0}(\alpha) + C_{l\,dail}(\alpha, \delta_{ail}) + C_{l\,drud}(\alpha, \delta_{rud}) + C_{l\,damp} \quad (C.30)$$

$$C_l = C_{l_0}(\alpha) + C_{l\,dail}(\alpha, \delta_{ail}) + C_{l\,drud}(\alpha, \delta_{rud}) + C_{l\,damp} \quad (C.31)$$

$$C_m = C_{m_0}(\alpha) + C_{m\,dele}(\alpha, \delta_{ele}) + C_{m\,damp} \quad (C.32)$$

$$C_n = C_{n_0}(\alpha) + C_{n\,dail}(\alpha, \delta_{ail}) + C_{n\,drud}(\alpha, \delta_{rud}) + C_{n\,damp} \quad (C.33)$$

$$C_{Y\,damp} = \frac{b}{2V} (C_{Y_p}(\alpha)p + C_{Y_r}(\alpha)r) \quad (C.34)$$

$$C_{l\,damp} = \frac{b}{2V} (C_{l_p}(\alpha)p + C_{l_r}(\alpha)r) \quad (C.35)$$

$$C_{m\,damp} = \frac{\bar{c}}{2V} (C_{m_q}(\alpha)q) \quad (C.36)$$

$$C_{n\,damp} = \frac{b}{2V} (C_{n_p}(\alpha)p + C_{n_r}(\alpha)r) \quad (C.37)$$

$$L = \bar{q} S C_L \quad (C.38)$$

$$D = \bar{q} S C_D \quad (C.39)$$

$$Y = \bar{q} S C_Y \quad (C.40)$$

$$l_{aero} = \bar{q} S b C_l \quad (C.41)$$

$$m_{aero} = \bar{q} S \bar{c} C_m \quad (C.42)$$

$$n_{aero} = \bar{q} S b C_n \quad (C.43)$$

$$F_{x\,aero} = L \sin \alpha - D \cos \alpha \quad (C.44)$$

$$F_{y\,aero} = Y \quad (C.45)$$

$$F_{z\,aero} = -L \cos \alpha - D \sin \alpha \quad (C.46)$$

**Table C.1. Polynomial Fit for Morphing Aircraft Dynamics (Loiter Configuration)**

| <b>Coefficient</b> | <b>Polynomial Fit</b>                                                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $C_{L_0}$          | $109.2699 \times 10^{-3} \cdot \alpha + 123.8907 \times 10^{-3}$                                                                                               |
| $C_{L\,dele}$      | $14.4800 \times 10^{-3} \cdot \delta_{ele} - 244.4444 \times 10^{-6}$                                                                                          |
| $C_{D_0}$          | $584.2330 \times 10^{-6} \cdot \alpha^2 + 1.2702 \times 10^{-3} \cdot \alpha + 18.1797 \times 10^{-3}$                                                         |
| $C_{D\,dele}$      | $53.1775 \times 10^{-6} \cdot \delta_{ele}^2 + 511.9082 \times 10^{-6} \cdot \delta_{ele} + 16.5974 \times 10^{-6}$                                            |
| $C_{D\,drud}$      | $18.6182 \times 10^{-6} \cdot \delta_{rud}^2 + 266.6667 \times 10^{-9} \cdot \delta_{rud} + 7.5758 \times 10^{-6}$                                             |
| $C_{Y_0}$          | 0                                                                                                                                                              |
| $C_{Y\,dail}$      | 0 (Approximation: assume $\beta \approx 0$ )                                                                                                                   |
| $C_{Y\,drud}$      | 0 (Approximation: assume $\beta \approx 0$ )                                                                                                                   |
| $C_{Y_p}$          | 0 (Approximation: assume $\beta \approx 0$ )                                                                                                                   |
| $C_{Y_r}$          | 0 (Approximation: assume $\beta \approx 0$ )                                                                                                                   |
| $C_{l_0}$          | 0 (Approximation: assume $\beta \approx 0$ )                                                                                                                   |
| $C_{l\,dail}$      | $-24.5948 \times 10^{-9} \cdot \delta_{ail}^2 - 5.6908 \times 10^{-3} \cdot \delta_{ail} + 3.5151 \times 10^{-6}$                                              |
| $C_{l\,drud}$      | $82.1813 \times 10^{-6} \cdot \delta_{rud} + 82.6667 \times 10^{-9}$                                                                                           |
| $C_{l_p}$          | $-87.4541 \times 10^{-6} \cdot \alpha^2 - 613.6533 \times 10^{-6} \cdot \alpha - 436.6504 \times 10^{-3}$                                                      |
| $C_{l_r}$          | $-102.5071 \times 10^{-6} \cdot \alpha^2 + 26.1115 \times 10^{-3} \cdot \alpha + 76.5307 \times 10^{-3}$                                                       |
| $C_{m_0}$          | $-29.6053 \times 10^{-3} \cdot \alpha - 73.3744 \times 10^{-3}$                                                                                                |
| $C_{m\,dele}$      | $-28.9819 \times 10^{-3} \cdot \delta_{ele} + 519.2222 \times 10^{-6}$                                                                                         |
| $C_{m_q}$          | $4.2263 \times 10^{-3} \cdot \alpha^2 - 2.8147 \times 10^{-3} \cdot \alpha - 27.5537$                                                                          |
| $C_{n_0}$          | 0                                                                                                                                                              |
| $C_{n\,dail}$      | $-81.1569 \times 10^{-9} \cdot \delta_{ail}^3 + 200.0000 \times 10^{-12} \cdot \delta_{ail}^2 + 115.9920 \times 10^{-6} \cdot \delta_{ail} - 1.3333 \times 10$ |
| $C_{n\,drud}$      | $-2.2077 \times 10^{-3} \cdot \delta_{rud} + 1.7209 \times 10^{-6}$                                                                                            |
| $C_{n_p}$          | $30.6181 \times 10^{-6} \cdot \alpha^2 - 4.9766 \times 10^{-3} \cdot \alpha - 19.9727 \times 10^{-3}$                                                          |
| $C_{n_r}$          | $-441.1887 \times 10^{-6} \cdot \alpha^2 - 790.4002 \times 10^{-6} \cdot \alpha - 210.0840 \times 10^{-3}$                                                     |

**Table C.2. Limits for Morphing Aircraft Aerodynamics**

| Variable       | Lower Limit (degrees) | Upper Limit (degrees) |
|----------------|-----------------------|-----------------------|
| $\alpha$       | -3                    | 15                    |
| $\delta_{ail}$ | -20                   | 20                    |
| $\delta_{ele}$ | -15                   | 5                     |
| $\delta_{rud}$ | -10                   | 10                    |

### C.3 Thrust and Force Transfer

The total body-axis forces and moments are calculated by moving the aerodynamic forces and moments to the aircraft center of gravity (C.G.) and adding in the thrust force from the engine. The engine is assumed to align with the body x-axis; hence the thrust force is added to  $F_x$  exclusively.

$$F_x = T + F_{x\text{aero}} \quad (\text{C.47})$$

$$F_y = F_{y\text{aero}} \quad (\text{C.48})$$

$$F_z = F_{z\text{aero}} \quad (\text{C.49})$$

$$l = l_{\text{aero}} \quad (\text{C.50})$$

$$m = F_{z\text{aero}} dx + m_{\text{aero}} \quad (\text{C.51})$$

$$n = F_{y\text{aero}} dx + n_{\text{aero}} \quad (\text{C.52})$$

### C.4 Transition Controller Design

For the purposes of transition control law design, the procedure developed by [Seto 1999] has been adopted here. For this architecture, the vehicle dynamics are represented by the following nonlinear equation:

$$\dot{x}(t) = f(x(t), u(t), t) \quad (\text{C.53})$$

In this model,  $x$  represents the  $n$ -dimensional state of the vehicle and  $u$  represents the  $m$ -dimensional control input. Both the state and the control input may be subject to constraints:

$$q_1(x) \leq 0, q_2(x) \leq 0, \dots q_j(x) \leq 0, j < n \quad (\text{C.54})$$

$$p_1(u) \leq 0, p_2(u) \leq 0, \dots p_r(u) \leq 0, r < m \quad (\text{C.55})$$

If the constraints given in Eq. (C.54)-(C.55) fully characterize the necessary requirements for safe operation of the vehicle, run-time assurance can be achieved if there exists a safety controller that can control the vehicle without violating the specified constraints. A discussion of the necessary constraints is the focus of the next sub-section. For the purposes of RTA design, we denote this safety controller the ‘transition’ controller. When it is detected that the advanced controller has failed, control is passed to the transition controller. The control objective of the

transition controller is to bring the system back to a state where the baseline controller can be used, without violating the constraints defined in Eq. (C.54)-(C.55). To enable definition of the RTA switching condition, it is necessary to define the restricted operational region of the transition controller. The simplex architecture can then be described as follows: monitor the state of the vehicle when the advanced controller is active; if the state reaches the boundary of restricted operational region of the transition controller, disengage the advanced controller and engage the transition controller. We borrow the definition of the operational region of a control law from [Seto 1998]:

**Definition:** Consider the plant in Eq. (C.53)-(C.55). Let the set of admissible states be defined as  $F = \{x : q_1(x) \leq 0, q_2(x) \leq 0, \dots, q_j(x)\}$ . Let the set of admissible controls be defined as  $\Omega = \{u : p_1(u) \leq 0, p_2(u) \leq 0, \dots, p_r(u)\}$ . An operational region (OR) for a given control law  $u$ , which takes values from  $\Omega$ , is defined as a subset  $O_u \subseteq F$ , such that under the control of  $u$ , the trajectory of the plant, starting from any state in  $O_u$ , will remain in  $O_u$  and satisfy the control objective of  $u$ .

The OR, however, may not be sufficient for this purpose in digital control, where one sampling period delay of control is inevitable. To take the sampling period into account, we define a *restricted operational region* (ROR) as follows. Let  $T$  be the sampling period of the system and  $\phi_u(t_o, x_o, t)$  be the solution of Eq. (C.53) at  $t > t_o$  with  $u$  taking values from  $\Omega$  and  $(t_o, x_o)$  the initial condition. Then a ROR  $R_u$  of the control law  $u$  is defined as a subset of  $O_u$  such that:

$$R_u = \{x : x \in O_u, \phi_v(t_o, x_o, t_o + T) \in O_u, \forall t_o \geq 0, \forall v \in \Omega\} \quad (C.56)$$

The dynamics of the morphing vehicle used for the inner-loop RTA challenge problem are significantly slower than the control law update rate (0.01 s). Hence, it is assumed that the operational region of the transition controller is a good approximation for the restricted operational region of the transition controller (i.e. the effect of  $T$  is negligible).

Of course, if the ROR of the transition controller is too small, it will unnecessarily limit the operational region of the advanced controller. Hence, it is desirable to design a transition controller with a large ROR, so that the advanced controller is only disengaged when a true operational limit has been reached. It is assumed that during advanced controller design, an operational envelope is specified. If the advanced controller is well designed, this operational envelope will not violate any of the constraints specified in Eq. (C.54)-(C.55). Hence, at a minimum, it will be necessary for the ROR of the transition controller to include the operational envelope of the advanced controller subject to the constraints specified in Eq. (C.54)-(C.55).

The technique that is used to design the transition controller is to identify an equilibrium state for the vehicle and then develop a control law that stabilizes the vehicle around the equilibrium state. It is clear that the stability region of the vehicle using this control law meets the definition of the operational region. This stabilization problem can be solved using techniques derived from Lyapunov stability theory. Except for a small subclass of systems, most nonlinear stabilization problems do not have known analytic solutions. The approach used to sidestep this issue is to linearize the nonlinear system at an equilibrium state, and then design and analyze the transition

controller using the local linear system. Specifically, let  $\delta x = x - x_e$  and  $\delta u = u - u_e$ . The local linear system is then given by the first order terms of the Taylor series expansion of the function  $f(x, u)$  around the equilibrium point  $(x_e, u_e)$ :

$$\delta \dot{x} = A\delta x + B\delta u \quad (C.57)$$

$$A = \left. \frac{\partial f(x, u)}{\partial x} \right|_{x=x_e} \quad (C.58)$$

$$B = \left. \frac{\partial f(x, u)}{\partial u} \right|_{x=x_e, u=u_e} \quad (C.59)$$

During the design of a transition controller,  $A$  and  $B$  are considered to be constant matrices. Hence, in order to build up a sufficiently large ROR, the transition controller will be composed of many different linear controllers each operating around its own equilibrium point. The ROR of the overall transition controller will then be the union of the RORs of each individual linear controller. This technique closely parallels the gain-scheduling laws that are currently used in certified production flight control systems.

#### C.4.1 Guarantee of Safe Operation

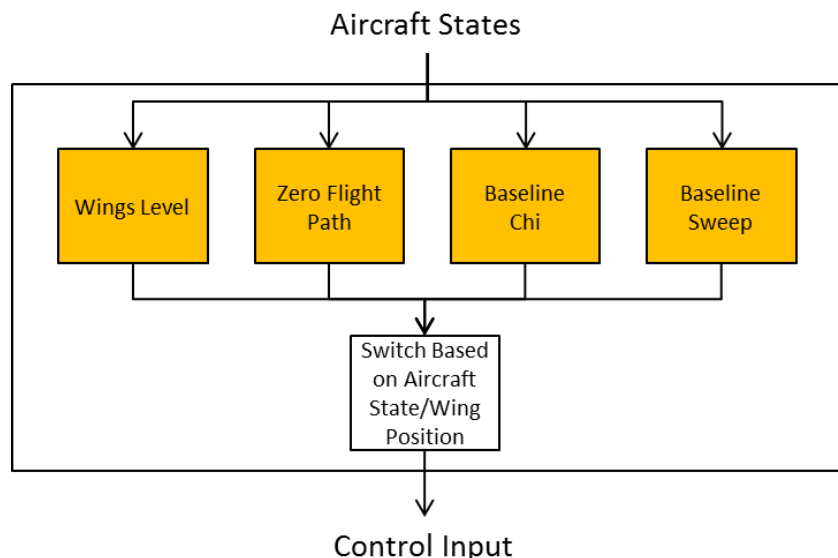
In the previous sub-section, we discussed that the ROR of the transition controller is designed to include the operational envelope of the advanced controller subject to the constraints specified in Eq. (C.54)-(C.55). This ensures that the RTA architecture does not unnecessarily limit the advanced controller. To ensure safe operation, we must now outline the constraints. Four classes of constraints are considered during RTA design:

1. Actuator Limits: Rate and saturation limits of the aerodynamic and morphing actuators.
2. Engine Limits: Maximum thrust provided by the engine.
3. Structural Limits: Maximum load factor and airspeed that can be handled by the airframe.
4. Modeling Limits: Region of the state-space where the aerodynamic and other relevant models are accurate.

The limits can be transformed into constraints on the state and control input in the form required by Eq. (C.54)-(C.55). The transition controller will be designed explicitly taking these constraints into consideration.

#### C.4.2 Transition Controller Architecture

The transition controller will follow the procedure depicted in Figure C.3 when the RTA switch is engaged. The architecture partitions the control task into a lateral stabilization component and a longitudinal stabilization component. This partition is commonly used in certified production flight control systems. Of course, the architecture must also address the added complexity of transitioning the wing back to the baseline configuration.



**Figure C.3. Transition Controller Architecture**

1. Go to 'Wings Level'. Perform lateral stabilization at the current wing configuration. The states of interest for this subsystem are: roll rate, yaw rate, sideslip, and roll angle. The outputs of this system are commands to the aileron and rudder. The wing is commanded to maintain the configuration that was active when the RTA switch was engaged.
2. Go to 'Zero Flight-Path'. Perform longitudinal stabilization at the current wing configuration. The states of interest for this subsystem are: pitch rate and angle-of-attack. The output of this system is a command to the elevator. The wing is commanded to maintain the configuration that was active when the RTA switch was engaged.
3. Go to 'Baseline Chi'. Transition to the wing to be value of chi that was used during baseline controller design, while maintain straight-and-level flight. The states of interest for this subsystem are: pitch rate and angle-of-attack. The output of this system is a command to the elevator and the wing actuator that changes chi.
4. Go to 'Baseline Sweep'. Transition to the wing to be value of sweep that was used during baseline controller design, while maintain straight-and-level flight. The states of interest for this subsystem are: pitch rate and angle-of-attack. The output of this system is a command to the elevator and the wing actuator that changes sweep.

While Stages 2-4 are primarily longitudinal actions, it will be necessary to continue lateral stabilization to account for any external disturbances. The ROR required for Stage 1 will be significantly larger than the ROR required for Stages 2-4, as it will only be necessary to ensure the transition controller has sufficient control authority to perform disturbance rejection for Steps Stages 2-4. Hence, whatever controller is used to perform lateral stabilization for Stage 1 can be used to perform lateral stabilization for Stages 2-4.



It will also be necessary to maintain airspeed control during the transition. The airspeed dynamics are significantly slower than the dynamics of the lateral and longitudinal states that are of interest to the transition controller. As a result, the airspeed control loop will be largely a feed-forward loop that outputs the thrust required to maintain straight-and-level flight at pre-specified airspeed. A linear feedback law will provide disturbance rejection.

### C.4.3 Equilibrium Analysis

The first step in transition control law design is to identify an equilibrium point. Equilibrium points are found by finding a straight-and-level (zero bank angle, zero flight-path angle) trim condition. In order to find a trim condition, a nonlinear set of equations is solved for angle-of-attack, thrust, and aerodynamic surface deflection. The equations are:

$$C_l = 0 \quad (C.60)$$

$$C_m = 0 \quad (C.61)$$

$$C_n = 0 \quad (C.62)$$

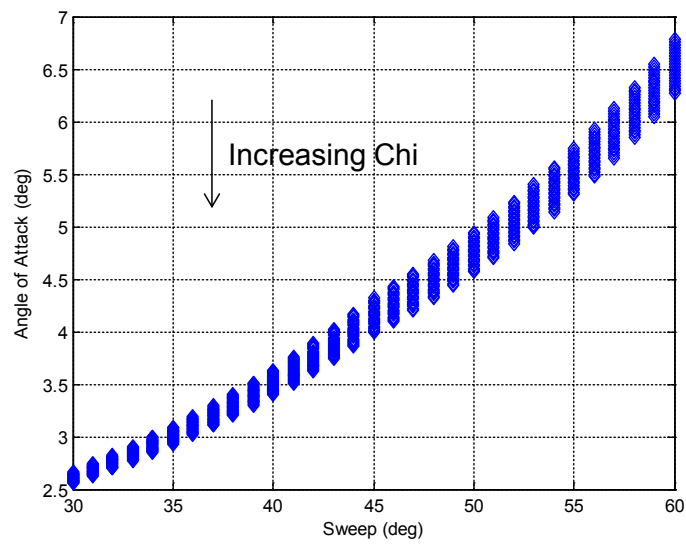
$$\bar{q}S_{ref}C_L + T_x \sin \alpha - mg = 0 \quad (C.63)$$

$$\bar{q}S_{ref}C_D + T_x \cos \alpha = 0 \quad (C.64)$$

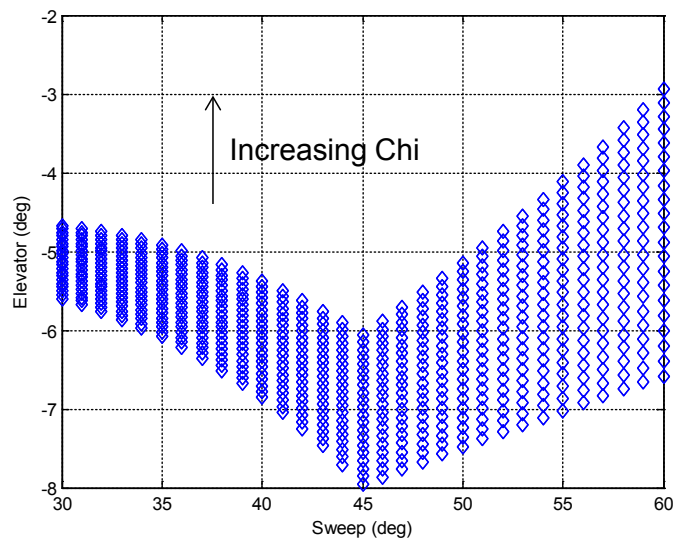
In these equations,  $m$  is the mass,  $T_x$  is the thrust along the body x-axis,  $S_{ref}$  is the wing reference area,  $C_L$  is the lift coefficient,  $C_D$  is the drag coefficient,  $C_l$  is the rolling moment coefficient,  $C_m$  is the pitching moment coefficient, and  $C_n$  is the yawing moment coefficient. Implicit in these equations are a number of parameters that define the trim condition. These include:

1. Airspeed (velocity in  $\bar{q}$ )
2. Altitude (air density in  $\bar{q}$ )
3. Wing configuration ( $C_L, C_D, C_l, C_m, C_n$ )
4. Vehicle attributes (mass, center-of-gravity, etc.)

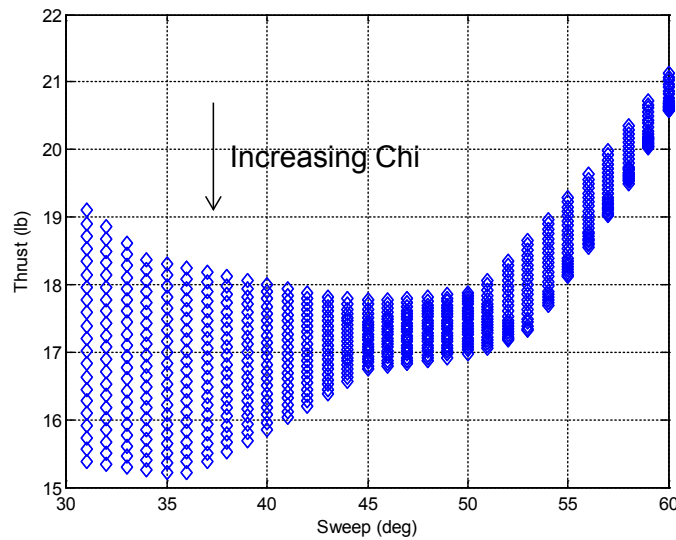
For the experiments conducted under this effort, the vehicle parameters are assumed to be fixed. A low altitude flight condition of 1500 feet has been selected as the focus. The following plots show the effect of airspeed and wing configuration on the trim condition. These parameters have the most pronounced effect on the trim condition considering the operational envelope of the vehicle. In the first set of plots, the vehicle was trimmed to an airspeed of 110 knots at an altitude of 1500 feet. Figure C.4 plots the trim angle-of-attack as a function of the two wing degrees-of-freedom (sweep and chi). Each diamond represents a single wing configuration. Inspecting this plot, as sweep increases, the trim angle-of-attack also increases. As chi increases, the trim angle-of-attack decreases. Figure C.5 plots the trim elevator deflection as a function of the two wing degrees-of-freedom. Inspecting this plot, as sweep increases, the trim elevator deflection initially decreases but then increases after about 45 degrees of sweep. As chi increases, the trim elevator deflection increases. Figure C.6 plots the trim thrust as a function of the two wing degrees-of-freedom. Inspecting this plot, as sweep increases, the trim thrust is somewhat flat but then increases after about 50 degrees of sweep. As chi increases, the trim thrust decreases.



**Figure C.4. Trim Angle-of-Attack (V=110 knots, h=1500 feet)**

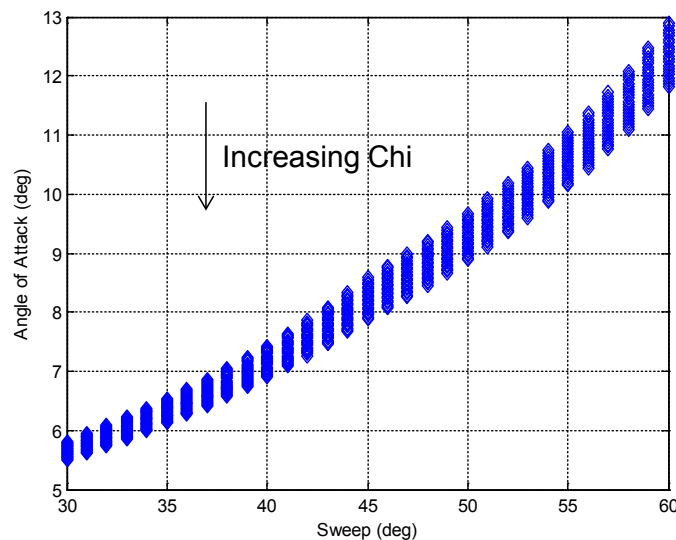


**Figure C.5. Trim Elevator Deflection (V=110 knots, h=1500 feet)**

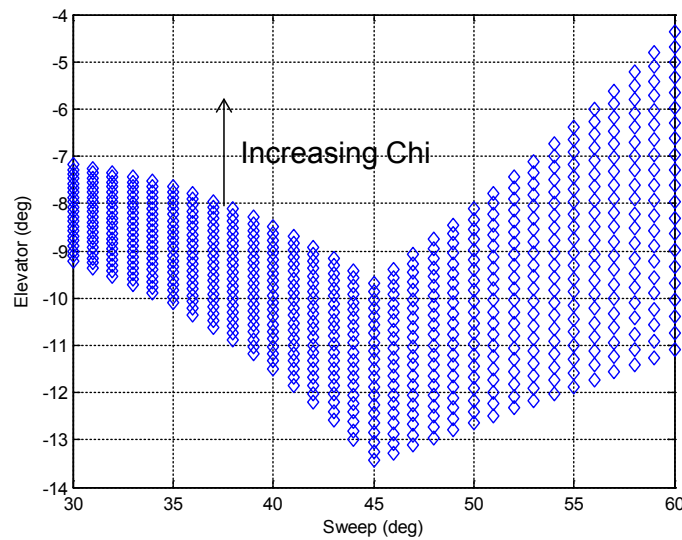


**Figure C.6. Trim Thrust (V=110 knots, h=1500 feet)**

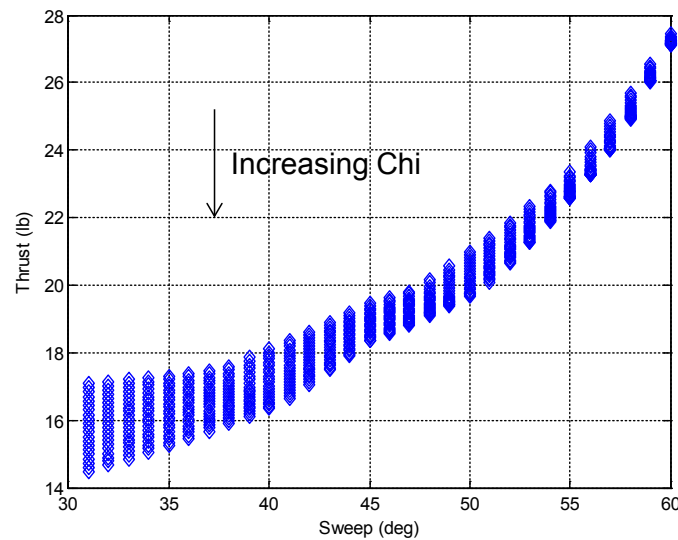
In the second set of plots, the vehicle was trimmed to an airspeed of 80 knots at an altitude of 1500 feet. The trends are consistent with the first set of plots. The most pronounced difference is in Figure C.9. The relationship between trim thrust and sweep is different with trim thrust increasing steadily as sweep increases. It is clear that for a lower airspeed, the trim angle-of-attack, trim thrust, and trim elevator deflection are all larger in magnitude (the trim elevator deflection is more negative).



**Figure C.7. Trim Angle-of-Attack (V=80 knots, h=1500 feet)**



**Figure C.8. Trim Elevator Deflection (V=80 knots, h=1500 feet)**



**Figure C.9. Trim Thrust (V=80 knots, h=1500 feet)**

#### **C.4.4 Stages 1-2: 'Wings Level', 'Zero Flight Path'**

In order to design a stabilizing control law to bring the vehicle to wings level, it is first necessary to isolate the appropriate dynamics. The lateral-directional dynamics linearized about the equilibrium point specified by a straight-and-level trim condition are given by the equations of motion for sideslip, bank angle, roll rate, and yaw rate:

$$A = \begin{bmatrix} \frac{Y_\beta}{V_t} & \frac{g \cos \theta_{trim}}{V_t} & \frac{Y_p}{V_t} & \frac{Y_r}{V_t} - 1 \\ 0 & 0 & \frac{1}{\cos \theta_{trim}} & 0 \\ L'_\beta & 0 & L'_p & L'_r \\ N'_\beta & 0 & N'_p & N'_r \end{bmatrix} \quad (C.65)$$

$$B = \begin{bmatrix} \frac{Y_{\delta ail}}{V_t} & \frac{Y_{\delta rud}}{V_t} \\ 0 & 0 \\ L'_{\delta ail} & L'_{\delta rud} \\ N'_{\delta ail} & N'_{\delta rud} \end{bmatrix} \quad (C.66)$$

$$\delta x = \begin{bmatrix} \beta \\ \phi \\ p \\ r \end{bmatrix} \quad (C.67)$$

$$\delta u = \begin{bmatrix} \delta_{ail} \\ \delta_{rud} \end{bmatrix} \quad (C.68)$$

In Eq. (C.65) and (C.66), the primed moment derivatives are defined according to common convention (e.g., see [Stevens 1992]). These dynamics are usually decomposed into the dutch roll mode represented by a pair of complex conjugate poles, the faster roll subsidence mode represented by a real pole, and a the slower spiral model represented by a a real pole. The effect of the dutch roll mode is small when sideslip is close to zero. The spiral mode is usually much slower than the roll subsidence mode and most likely longer than the amount of time it should take to transition from the advanced controller to the baseline controller. Hence, initial investigations have focused on the roll subsidence mode. The lateral dynamics governed by the roll subsidence mode can be approximated by:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & L'_p \end{bmatrix} \quad (C.69)$$

$$B = \begin{bmatrix} 0 \\ L'_{\delta ail} \end{bmatrix} \quad (C.70)$$

$$\delta x = \begin{bmatrix} \phi \\ p \end{bmatrix} \quad (C.71)$$

$$\delta u = \delta_{ail} \quad (C.72)$$

A similar decomposition is performed for the longitudinal aircraft dynamics into the faster short-period mode and the slower phugoid mode. Both modes are represented by a pair of complex conjugate poles. The short-period mode involves the angle-of-attack and pitch rate of the vehicle, where the phugoid mode involves airspeed and pitch angle. The short-period mode is usually a heavily damped oscillation with a period of a few seconds. The period is short enough so airspeed does not change by any significant amount. The phugoid mode has a nearly constant angle-of-attack but varying pitch, caused by a repeated exchange of airspeed and altitude. As airspeed decreases, the nose drops causing airspeed to increase again. As airspeed increases, the nose rises again and so on in a cyclic manner. The period of the phugoid mode is much longer than the amount of time it should take to transition from the advanced controller to the baseline controller. Hence, the transition controller will focus on the short-period mode. The short-period dynamics linearized about the equilibrium point specified by a straight-and-level trim condition are given by the equations of motion for angle-of-attack and pitch rate:

$$A = \begin{bmatrix} \left( \frac{\bar{q}S}{mV_T} \right) C_{L,\alpha} & 1 \\ \left( \frac{\bar{q}S\bar{c}}{I_{yy}} \right) C_{m,\alpha} & \left( \frac{\bar{q}S\bar{c}}{I_{yy}} \right) \left( \frac{\bar{c}}{2V_T} \right) C_{m,q} \end{bmatrix} \quad (C.73)$$

$$B = \begin{bmatrix} \left( \frac{\bar{q}S}{mV_T} \right) C_{L,\delta_{ele}} \\ \left( \frac{\bar{q}S\bar{c}}{I_{yy}} \right) C_{m,\delta_{ele}} \end{bmatrix} \quad (C.74)$$

$$\delta x = \begin{bmatrix} \alpha - \alpha_{trim} \\ q \end{bmatrix} \quad (C.75)$$

$$\delta u = \delta_{ele} - \delta_{ele,trim} \quad (C.76)$$

Note that the trim pitch rate is zero. As mentioned earlier, the  $A$  and  $B$  matrices are parameterized by wing configuration, airspeed, and altitude. With the local linear system identified, it is now possible to use the techniques outlined in [Seto 1999] to design the Stage 1 and Stage 2 transition controllers. To reduce the number of transition controllers, it is desirable to maximize the stability region of each controller. The linear matrix inequality (LMI) problem given in Eq. (C.77) is formulated to accomplish this goal. Let the stability region be defined as  $S = \{x : x^T P x \leq 1\}$ . The volume of this region is proportional to  $\sqrt{\det P^{-1}}$ . Let  $Q = P^{-1}$ ; it is clear as  $\det Q^{-1}$  decreases, the volume of the stability region increases.

$$\begin{aligned}
& \min \quad \log \det Q^{-1} \\
& s.t. \quad Q > 0; \\
& \quad \quad QA^T + AQ + Z^T B^T + BZ < 0; \\
& \quad \quad a_i^T Q a_i \leq 1, \quad i = 1, \dots, l; \\
& \quad \quad \begin{bmatrix} 1 & b_j^T Z \\ Z^T b_j & Q \end{bmatrix} \geq 0, \quad j = 1, \dots, r
\end{aligned} \tag{C.77}$$

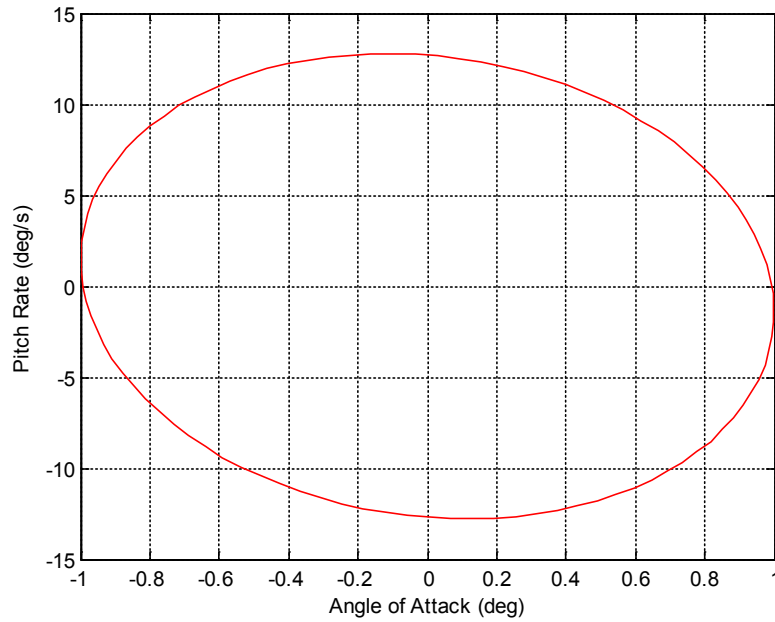
In this formulation,  $a_i^T \delta x \leq 1$  defines the constraints on the state and  $b_j^T \delta u \leq 1$  defines the constraints on the control input. Because a local linear system is used to develop the stabilizing controller for a nonlinear system, it is necessary to limit  $\delta x$  and  $\delta u$  to be small enough perturbations from the equilibrium point to ensure the validity of the local linear approximation. As a result, the constraints defined in Eq. (C.54)-(C.55) may not be sufficient to guarantee safe operation of the transition control law. Tighter constraints will be required to ensure that the stability region calculated for the local linear system is a good approximation of the stability region of the nonlinear system. The linear feedback law is calculated from the solution to this problem as:  $K = ZQ^{-1}$ ,  $\delta u = K\delta x$ . The stability region of this feedback law will be the largest region specified by a quadratic Lyapunov function with respect to all possible linear feedback laws that render asymptotic stability for the local linear system.

As an example, an equilibrium point is generated for a straight-and-level flight condition with a wing configuration of ( $S = 45$ ,  $\chi = 50$ ), an airspeed of 190 ft/s, and an altitude of 1500 ft. Figure C.10 shows the stability region of the linear feedback law that results from the solution of the LMI problem in Eq. (C.77). The following constraints are used to develop the stabilizing control law:

$$-1 \leq \delta \alpha \leq 1 \tag{C.78}$$

$$-1 \leq \delta \delta_{ele} \leq 1 \tag{C.79}$$

Because the equilibrium point is far away from physical and design limits, the constraints are used purely to ensure agreement between the local linear system and the nonlinear system. As the aerodynamic increments are a function angle-of-attack and elevator deflection, it makes sense to enforce limits on these variables. At this stage of design, a measure of the agreement between the two systems has not been formally enforced.



**Figure C.10. Stability Region of Linear Feedback Law**

For a given linear feedback law, it is also useful to ascertain the largest region specified by a quadratic Lyapunov function that renders asymptotic stability. This stability region can be found by solving the LMI problem given in Eq. (C.80).

$$\begin{aligned}
 \min \quad & \log \det Q^{-1} \\
 \text{s.t.} \quad & Q > 0; \\
 & Q\bar{A}^T + \bar{A}Q < 0; \\
 & \alpha_k^T Q \alpha_k \leq 1, \quad k = 1, \dots, p;
 \end{aligned} \tag{C.80}$$

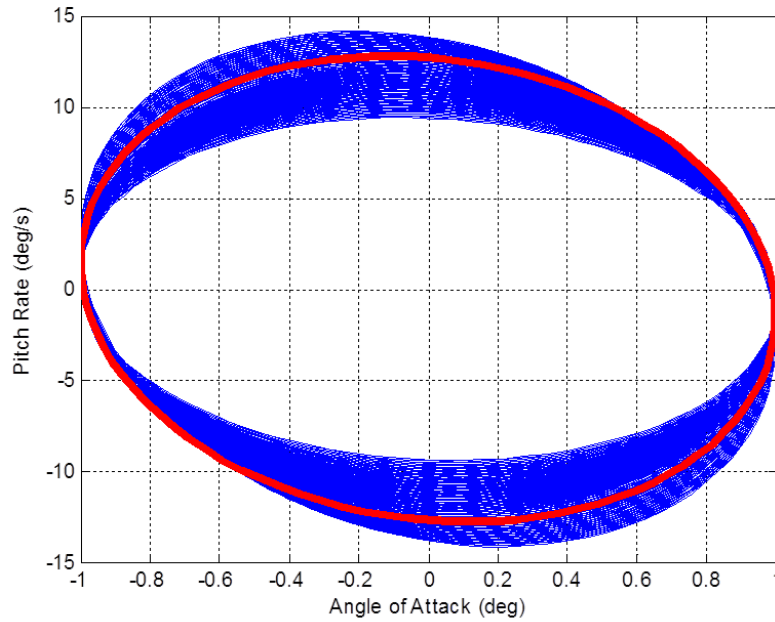
In this equation,  $\bar{A} = A + BK$  is the closed-loop system and  $\alpha_k = a_k$ ,  $k = 1 \dots l$ ,

$\alpha_k = b_j^T K$ ,  $j = 1 \dots r$ ,  $k = l + j$ , encapsulates the constraints on both the state and the control input.

By performing this analysis, it is possible to gain insight into how changing the equilibrium point and the local linear system affects the shape of the stability region. Equilibrium points and local linear systems are generated for multiple wing configurations with sweep angles varying between 30 and 60 degrees and chi angles varying between 30 and 50 degrees

( $S = [30; 60]$ ,  $\chi = [30; 50]$ ), at an airspeed of 190 ft/s and an altitude of 1500 ft. The LMI problem in Eq. C.80 is solved for each of these systems using the linear feedback law that was optimized for ( $S = 45$ ,  $\chi = 50$ ). Figure C.11 plots the results of this analysis. It is clear from this plot that the area of the ellipsoid changes significantly, increasing as much as 9% over the ( $S = 45$ ,  $\chi = 50$ ) value, and decreasing as much as 26%. From this demonstration, we can see the importance of parameterizing the transition controller design by wing configuration to guarantee that adequate coverage is achieved.





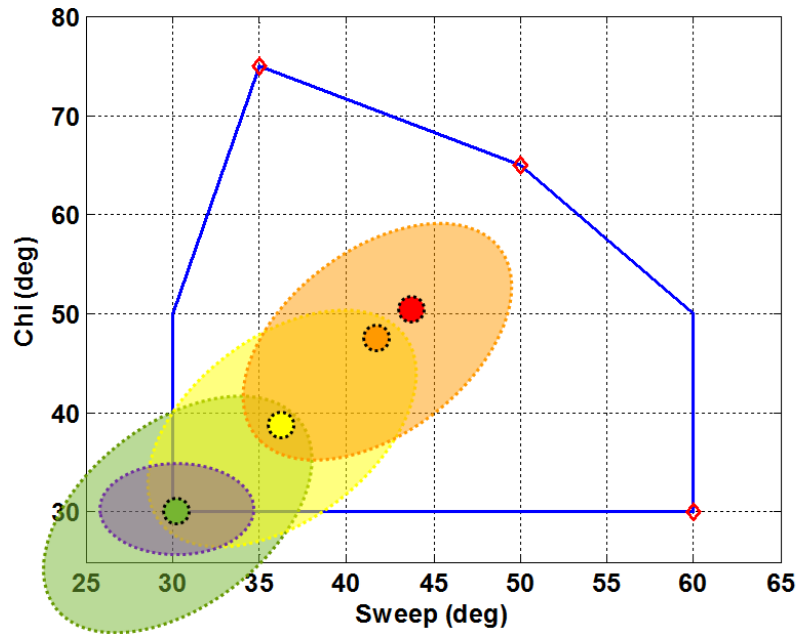
**Figure C.11. Stability Region for Various Equilibrium Points**

#### **C.4.5 Stages 3-4: Baseline Sweep and Baseline Chi**

The design of the Stage 3 and 4 of the transition controller will also focus on the longitudinal dynamics outlined in Eq. (C.73)-(C.76). The LMI problem outlined in Eq. (C.77) is also used to develop the linear feedback laws that comprise Stage 3 and 4 of the transition controller. While the goal of the Stage 2 is to stabilize the longitudinal states around a straight-and-level flight, the goal of the Stage 3 and 4 controllers is to morph the wing to the baseline configuration while maintaining straight-and-level flight. As a result, during Stage 3 and 4, the equilibrium point will change as the wing configuration changes. The Stage 3 and 4 controllers rely on the following process to ensure that the baseline wing configuration is achievable:

Select a setpoint  $r_k$  such that  $r_k$  is an exponentially stable equilibrium point for transition controller  $C_k$  and an  $\varepsilon$ -ball of the equilibrium point is in the region of attraction of transition controller  $C_{k+1}$ .

In [Bak 2013], it has been shown that a control law that is designed using this process provides a guarantee of progress (e.g. the control law will converge upon the desired endpoint within finite time). This guarantee of progress is important to ensure that at the end of Stage 4, the baseline wing configuration will be achieved. Figure C.12 provides a notional example of how the linear feedback laws in Stage 3 and 4 will be assembled to achieve the guarantee of progress.



**Figure C.12. Notional Example Depicting Guarantee of Progress**

In this figure, assume that at the end of Stage 2, the wing is at the configuration denoted by the red circle. Assume the ellipses are the ‘regions of attractions’ or stability regions for each individual controller. The red circle is within the region of attraction of the orange controller ( $C_1$ ). The equilibrium point of the orange controller is the orange dot. Hence, the first setpoint ( $r_1$ ) will be the orange dot. The orange dot is within the region of attraction of the yellow controller ( $C_2$ ). The equilibrium point of the yellow controller is the yellow dot. Hence, the second setpoint ( $r_2$ ) will be the yellow dot. The yellow circle is within the region of attraction of the green controller ( $C_3$ ). The equilibrium point of the green controller is the green dot. Hence, the third setpoint ( $r_3$ ) will be the green dot. The green dot is within the region of attraction of the baseline controller depicted by the grey ellipse. At this point, the transition is complete, and the baseline controller takes control of the vehicle.

In Figure C.12 the stability regions are depicted as two-dimensional ellipses over the sweep-chi space. In reality, the stability regions are multi-dimensional ellipsoids that are parameterized by wing configurations, as well as airspeed, altitude, and any other vehicle parameters that affect the equilibrium point. The setpoint  $r_k$  is a command that encapsulates all of these states.

## Appendix D

### Unmanned Aircraft Systems 3-DOF Dynamic Modeling

#### D.1 UAS Category Description

This chapter reviews dynamic modeling of UASs for use in our mission management level challenge problem. We first begin with a review of the general components that make up the control, guidance, navigation and mission feedback loops. Much literature on this subject exists, and one recommended text can be found in [Beard 2011]. UASs come in a variety of sizes and mission capabilities. Traditionally they have been organized into different tier classes. Although each branch of the military has their own tier definitions, they are all similar in how UASs are categorized. The following table presents the Air Force definition of UAS tiers [OSD 2005].

**Table D.1. UAS Tier Classes for the U.S. Air Force**

| <b>Tier Class</b> | <b>Description</b>                                                                                                                                                                                                                                                           |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Tier N/A          | Small/Micro UAS. Role filled by BATMAV (Wasp Block III).                                                                                                                                                                                                                     |
| Tier I            | Low altitude, long endurance. Role filled by the Gnat 750                                                                                                                                                                                                                    |
| Tier II           | Medium altitude, long endurance (MALE). Role currently filled by the MQ-1 Predator and MQ-9 Reaper.                                                                                                                                                                          |
| Tier II+          | High altitude, long endurance conventional UAS (or HALE UAS). Altitude: 60,000 to 65,000 feet (20,000 m), less than 300 knots (560 km/h) airspeed, 3,000-nautical-mile (5,600 km) radius, 24 hour time-on-station capability. Role currently filled by the RQ-4 Global Hawk. |
| Tier III          | High altitude, long endurance low-observable UAS. Same parameters as, and complementary to, the Tier II+ aircraft. The RQ-3 DarkStar was originally intended to fulfill this role.                                                                                           |






More recently, UASs have been categorized into groups, mainly for activities involving their integration into the National Airspace System (NAS). The following information and Table D.2 are repeated directly from [JUAS 2009] and [DoD 2011].

- Group 1: Typically hand-launched, self-contained, portable systems employed for a small unit or base security. They are capable of providing “over the hill” or “around the corner” reconnaissance and surveillance. They operate within visual range and are analogous to radio-controlled model airplanes as covered in AC 91-57.30.
- Group 2: Small to medium in size and usually support brigade and intelligence, surveillance, reconnaissance, and target acquisition requirements. They usually operate from unimproved areas and launched via catapult. Payloads may include a sensor ball with electro-optic / infrared (EO/IR) and laser range finder/designator (LRF/D) capability. They typically perform special purpose operations or routine operations within a specific set of restrictions.
- Group 3: Operate at medium altitudes with medium to long range and endurance. Their payloads may include a sensor ball with EO/IR, LRF/D, signal intelligence (SIGINT), communications relay, and chemical biological radiological nuclear explosive (CBRNE)

detection. They usually operate from unimproved areas and may not require an improved runway.

- **Group 4:** Relatively large UAS that operate at medium to high altitudes and have extended range and endurance. They normally require improved areas for launch and recovery, beyond line-of-sight (BLOS) communications, and have stringent airspace operations requirements. Payloads may include EO/IR sensors, radars, lasers, communications relay, SIGINT, Automatic Identification System (AIS), and weapons.
- **Group 5:** Include the largest systems, operate at medium to high altitudes, and have the greatest range, endurance, and airspeed capabilities. They require improved areas for launch and recovery, BLOS communications, and the most stringent airspace operations requirements. Group 5 UAS perform specialized missions such as broad area surveillance and penetrating attacks.

**Table D.2. UAS Group Definitions**

| UAS Groups | Maximum Weight (lbs) (MGTOV) | Normal Operating Altitude (ft) | Speed (kts)  | Representative UAS                                                      |                                                                                       |
|------------|------------------------------|--------------------------------|--------------|-------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| Group 1    | 0 – 20                       | <1200 AGL                      | 100          | Raven (RQ-11), WASP                                                     |    |
| Group 2    | 21 – 55                      | <3500 AGL                      | < 250        | ScanEagle                                                               |  |
| Group 3    | < 1320                       | < FL 180                       |              | Shadow (RQ-7B), Tier II / STUAS                                         |  |
| Group 4    | >1320                        |                                | Any Airspeed | Fire Scout (MQ-8B, RQ-8B), Predator (MQ-1A/B), Sky Warrior ERMP (MQ-1C) |  |
| Group 5    |                              |                                |              | Reaper (MQ-9A), Global Hawk (RQ-4), BAMS (RQ-4N)                        |  |

## D.2 UAS Sensor/Actuator Systems and Feedback Architectures

**Sensor systems:** Typical sensing equipment found on UASs are listed in Table D.3. However, note that not all UASs will have all sensors listed and other UASs may have more sensing equipment than listed - depending on what class of vehicle (e.g. smaller, cheaper UASs will most likely not have radar because of size and power requirements).

UAS processors will have sensor information conditioning capability since raw sensor data will typically have some noise/errors or drift. This conditioning will usually be in the form of some type of low pass filter or Kalman filter to give a smooth, optimal estimate of the parameter being sensed.

**Table D.3. Typical Sensor and Other Equipment for UASs**

| <b>Sensor</b>                         | <b>Description/Purpose</b>                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Inertial Measurement Unit (IMU)       | IMUs will be made up of 3-axis gyroscopes and accelerometers. IMU outputs are fed into a Inertial Navigation System (INS) where they are then integrated to get velocity, position, altitude and airframe attitude; typically primary sensor for inner-loop attitude control, outer-loop guidance, and short term navigation.                                                                                                                                      |
| Global Position System (GPS) Receiver | GPS may be used as the primary navigation sensor, but it also corrects for IMU integration drift. IMU measurements become primary when GPS signal is lost.                                                                                                                                                                                                                                                                                                         |
| Magnetometer                          | Used for heading measurement.                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Wind vanes/sideslip indicator         | For angle-of-attack (AOA) and angle-of-sideslip. Historically not typical equipment on UASs due to high cost and unreliable/noisy measurements. However, these are becoming standard sensors on fighter aircraft, and their costs may be reducing and reliability improving as improvements are made to these types of sensors. Note also, vehicles can be equipped with a sideslip indicator which uses a comparison of an accelerometer output to the IMU output |
| Pitot tube                            | Measure dynamic pressure: used for velocity measurements.                                                                                                                                                                                                                                                                                                                                                                                                          |
| Barometer                             | Measures static pressure: used for altitude and Mach number calculations.                                                                                                                                                                                                                                                                                                                                                                                          |
| Radar altimeter                       | Used for altitude measurements during approach to landing; usually only accurate to ~200 ft above ground level                                                                                                                                                                                                                                                                                                                                                     |
| Radar                                 | Used for situational awareness – collision avoidance, navigation, etc.                                                                                                                                                                                                                                                                                                                                                                                             |
| Optical devices                       | Forward looking infrared, electro-optical: used for situational awareness – collision avoidance, ISR, terrain mapping, etc.                                                                                                                                                                                                                                                                                                                                        |
| Radio transmitter / receiver          | Used to downlink data (camera images, position/velocity data, etc.), uplink commands from remote human operator or directions from command center.                                                                                                                                                                                                                                                                                                                 |

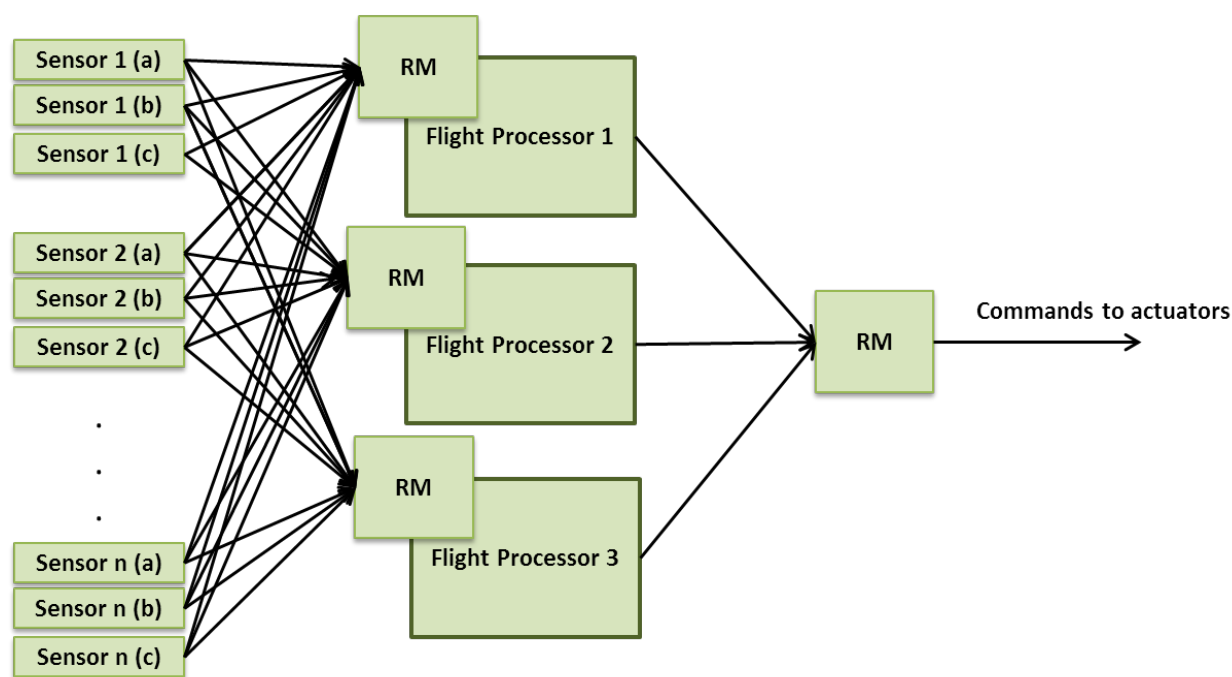
The onboard sensor packages are typically categorized into an air data system (ADS) and an inertial navigation system (INS). The air data system is comprised of the barometer, Pitot tube, and AOA and sideslip sensors. The INS is comprised of the IMU, GPS, magnetometer, radar altimeter and an onboard processor. It is also often referred to as the INS/GPS system. The INS takes in raw sensor inputs, filters these data and estimates current system states, usually using a Kalman filter. The states are current position, velocity, angular rates and vehicle attitude.

Future UASs will also be equipped with automatic dependent surveillance – broadcast (ADS-B). There are two parts to the ADS-B system: 1) “ADS-B out” takes in data from the INS/GPS and broadcasts its state to nearby aircraft and air traffic control (ATC) stations; 2) “ADS-B in” receives ADS-B out data that was broadcast by nearby aircraft. The goal is to provide situational awareness and allow self-separation.

**Redundancy management (RM)** – high value aircraft may have triple redundant sensors when possible. An RM algorithm will take in the three signals and compare their values and perform a voting scheme if the values differ. The most fundamental voting scheme is if one sensor has a different value than the other two, then it is assumed to be incorrect and the value of the other two is used. However, RM algorithms can be more complicated/sophisticated, using statistical methods on past measurements and filtering/estimation. This will be needed if all three signals are different or if there is only dual redundancy. Due to cost and space/weight considerations, not all sensors will have redundancy (e.g. there may be only one radar or one camera unit).

Also, triplex system architectures can get quite complex, as high value vehicles may have triple redundant flight critical processors, with possibly dual redundant mission critical processors – all having RM voting schemes, integrated with triplex sensor systems. Figure D.1 presents a general framework for such a multi-layered system.

High value UASs may also have redundant actuators if cost, space and weight considerations allow. Smart actuators will have a processor/sensors (e.g. potentiometers) housed with the unit to determine if the actuator actually does what it was commanded to do. It may be able to correct itself depending on how sophisticated the actuator is. Redundant actuator frameworks may all share the “load” in some fashion or there may be one primary actuator with secondary actuators only taking over if the primary actuator stops working.



**Figure D.1. Complex RM Systems with Multi-Layered Triplex Architectures**

Note that very high value aircraft (manned commercial and fighter jets) often have more than three channels for flight critical functions. These aircraft typically have at a minimum four flight critical processors – so the RM algorithms and multiplex framework are even more complicated.

**Control actuation systems** - Typical actuation/effector systems on a UAS are as follows:

1. Elevators: these effectors are flaps located on the rear horizontal tails and provide pitch moment control (nose up or down control).
2. Ailerons: these effectors are flaps located at the ends of the wings and provide roll moment control.
3. Rudder: these effectors are flaps located at the end of the rear vertical tail and provide yaw moment control (nose side to side control).
4. Inboard flaps: these effectors are flaps located on the inboard area of the wings (close to the fuselage) and are deployed at takeoff and landing to provide more lift at slower velocities (at the cost of higher drag). The flaps are retracted once the vehicle is airborne and in nominal operating condition.
5. Leading edge flaps: these effectors are flaps located at the leading edge of the wing and are usually deployed with the inboard flaps to provide more lift at slower velocities. Not all UAS configurations will necessarily employ leading edge flaps.
6. Spoilers: these effectors are located in the mid-section of the top of the wing and are deployed up to reduce lift over the wing. These effectors are used to descend in altitude during final approach to landing, or after touchdown to arrest any lift during rollout.
7. Speedbrakes: these effectors come in a variety of configurations, such as clamshell deployment of either the rudder, elevator, inboard flaps or ailerons or a combination of some or all of these effectors. Or, the speedbrake may be a separate flap located somewhere on the fuselage. Once deployed, its purpose is to add profile drag to the UAS configuration to help slow the forward speed of the vehicle.
8. Power plant: whether an electric powered or gas powered propeller engine or a turbofan jet engine, the UAS's power plant can be thought of as the velocity effector or actuator, providing the necessary power or thrust to follow the commanded or desired velocity.

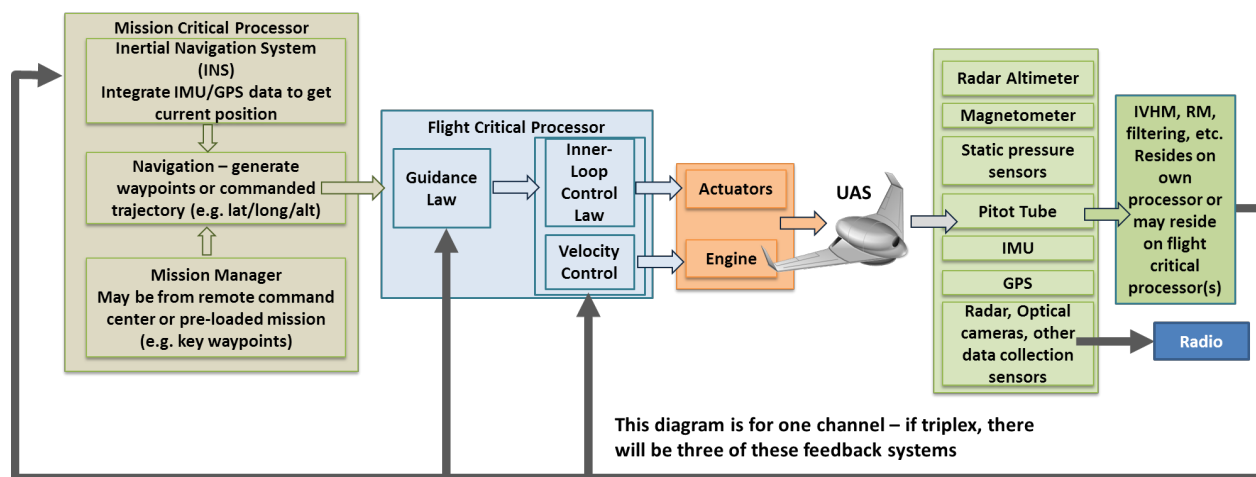
Note that redundant sensors and actuators cost money, weight and space. On small UASs, or UASs designed to be cheap and/or expendable, it is desired not to have all this redundancy. For this reason, there is wide interest in adaptive control algorithms, and fault detection, isolation and recovery (FDIR) (also known as fault tolerant systems with vehicle health management). These systems can adapt and reallocate control to recover flight stability. For example, yaw control can be recovered if the rudder stops working by using a new combination of ailerons and differential elevator deflections.

Figure D.2 presents an overall feedback system for a UAS. Note that this only shows one feedback channel. For a mixed duplex mission planning/triplex flight critical system, for example, there would be two and three nested parallel feedback channels. Furthermore, for each sensor shown in this figure, it itself could be dual or triply redundant. Figure D.3 shows an example of a triple redundant GPS receiver.

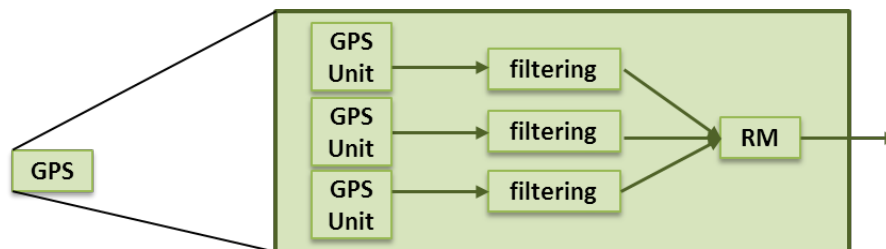
The overall feedback system has the following main feedback levels:

**Inner-Loop Control:** The inner-loop controller takes in commanded attitudes and determines commands to the actuators to smoothly follow the commanded attitude as well as maintain attitude stability. The inner-loop feedback system can be thought of as “orientation control” – it controls the {roll, pitch, yaw} to follow the desired path.

**Velocity Control:** The velocity controller takes in commanded velocity and generates either a commanded thrust, or commanded power. This is sometimes considered a part of the guidance system, sometimes a part of the control law. One key objective is to maintain the required speed so that the aircraft does not stall. The engine itself will have its own feedback controller which takes in the commanded thrust or power and typically delivers a fuel flow rate command to the engine to maintain or regulate the commanded thrust or power. For more advanced turbofan engines, the engine control system can be more complex, regulating and monitoring temperatures and pressures at various internal engine stages.



**Figure D.2. Overall Feedback System for a UAS**



**Figure D.3. Example of a Triple Redundant GPS Receiver**

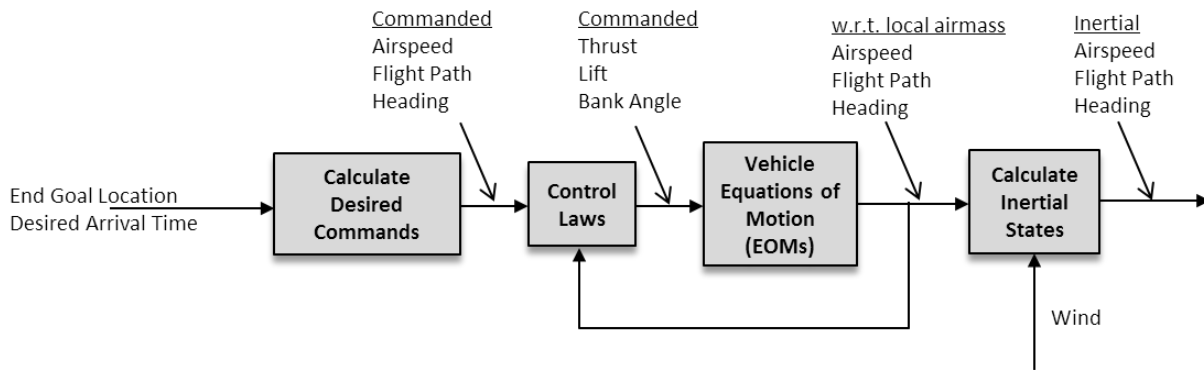
**Guidance Law:** The outer-loop guidance law takes in a commanded trajectory or waypoint and determines attitude commands such that the aircraft will follow the commanded path. There are several guidance modes, such as takeoff, landing (auto-land systems), cruise, dash, collision avoidance, automatic ground collision avoidance system (A-GCAS), etc. The guidance law can be thought of as “position control” – it controls the {x,y,z} path of the vehicle to get where the mission planning system wants the UAS to go.



Mission Management or Planning and Navigation: The mission planning function may interact with ground control operators, (or air vehicle operators (AVOs)) that upload desired paths or waypoints to follow. Or, such mission planning may be performed autonomously in more advanced, future systems that generate newly directed paths in real time to address changing environmental conditions, changing mission objectives due to new and unforeseen events, or other myriad unplanned causes. Mission management works integrally with the navigation system, which continually determines the vehicles current position and velocity. This information is then used to plan future paths to meet the most immediate objective, which can be considered a “sub-objective” in the overall mission flight plan. The mission management level is currently open to a wide array of design approaches as the UAS community is transitioning from strictly remotely piloted vehicles (RPVs) where all decisions requiring “intelligent, higher-level thought processes” are performed by the human operators and planners to more autonomous decision making based on sensed environmental conditions and acquired intelligence of the current unfolding mission scenario. Near future mission planning may be a combination of human interaction and autonomous functionality, alleviating the human flight manager from the lower level planning tasks so that they can concentrate on the larger, overall mission objectives.

### D.3 UAS Guidance Equations of Motion Model

The objective here is to develop a closed loop model of a UAS feedback system to be used for simulation for the mission management level RTA challenge problem development. The feedback system for this notional UAS model is shown in Figure D.4.



**Figure D.4. Feedback System for UAS Guidance Model**

#### Inertial and Local Air Mass States

Let the inertial velocity vector be given by

$$\vec{V}_i = \dot{X}_i \hat{i} + \dot{Y}_i \hat{j} - \dot{H}_i \hat{k} \quad (D.1)$$

as shown in Figure D.5. From the figure it can be seen that the inertial velocity vector components are given by

$$\begin{aligned}
\dot{X}_i &= V_i \cos(\gamma_i) \cos(\psi_i) \\
\dot{Y}_i &= V_i \cos(\gamma_i) \sin(\psi_i) \\
\dot{H}_i &= V_i \sin(\gamma_i)
\end{aligned}
\tag{D.2}$$

Where  $\gamma_i$ ,  $\psi_i$  define the inertial flight path and heading angles, respectively.



**Figure D.5. Inertial Frame of Reference for UAS**

From Eq. (D.2) the magnitudes of the inertial states can be derived and are given by

$$\begin{aligned}
V_i &= \sqrt{\dot{X}_i^2 + \dot{Y}_i^2 + \dot{H}_i^2} \\
\gamma_i &= \sin^{-1} \left( \frac{\dot{H}_i}{V_i} \right) = \tan^{-1} \left( \frac{\dot{H}_i}{\sqrt{\dot{X}_i^2 + \dot{Y}_i^2}} \right) \\
\psi_i &= \sin^{-1} \left( \frac{\dot{Y}_i}{V_i \cos(\gamma_i)} \right) = \tan^{-1} \left( \frac{\dot{Y}_i}{\dot{X}_i} \right)
\end{aligned}
\tag{D.3}$$

For keeping track of the current inertial quadrant, the inverse tangent function should be used in the above equations. For example, in MATLAB, one would use the atan2(Y,X) function to obtain the proper value for flight path and heading angles in all four quadrants. Let the local steady wind vector be:

$$\vec{W}_i = w_x \hat{i} + w_y \hat{j} - w_h \hat{k}
\tag{D.4}$$

And let the local velocity,  $V$ , be defined by the relation

$$V_i = V + W_i
\tag{D.5}$$

Then the inertial velocity components are given by

$$\begin{aligned}
\dot{X}_i &= V \cos(\gamma) \cos(\psi) + w_x \\
\dot{Y}_i &= V \cos(\gamma) \sin(\psi) + w_y \\
\dot{H}_i &= V \sin(\gamma) + w_h
\end{aligned}
\tag{D.6}$$

Where  $\gamma$  is the flight path angle and  $\psi$  is the heading angle, both w.r.t. the local air mass.

### UAS Guidance Equations of Motion

The vehicle's 3-DOF equations of motion w.r.t. the local air mass are given by

$$\begin{aligned}\dot{V} &= \frac{T-D}{M} - g \sin(\gamma), \quad D = C_{D1}V^2 + C_{D2} \frac{L^2}{V^2} \\ \dot{\gamma} &= \frac{L}{MV} \cos(\phi) - \frac{g}{V} \cos(\gamma) \\ \dot{\psi} &= \frac{L}{MV} \frac{\sin(\phi)}{\cos(\gamma)}\end{aligned}\tag{D.7}$$

where, the parameters are listed in Table D.4.

**Table D.4. Definition of Parameters in Equations of Motion**

| Parameter        | Definition                                                                                                                            |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| $g$              | Acceleration of gravity (ft/sec <sup>2</sup> )                                                                                        |
| $M$              | Total aircraft mass (slugs)                                                                                                           |
| $V$              | Airspeed w.r.t. local air mass (ft/sec)                                                                                               |
| $\gamma$         | Flight path angle w.r.t. local air mass (rad)                                                                                         |
| $\psi$           | Heading angle w.r.t. local air mass (rad)                                                                                             |
| $D$              | Total drag (lbs)                                                                                                                      |
| $C_{D1}, C_{D2}$ | Coefficients in drag equation, $C_{D1}$ in lb-sec <sup>2</sup> /ft <sup>2</sup> , $C_{D2}$ in ft <sup>2</sup> /(lb-sec <sup>2</sup> ) |
| $T$              | Total thrust (lbs)                                                                                                                    |
| $L$              | Total lift (lbs)                                                                                                                      |
| $\phi$           | Aircraft bank angle about the velocity vector w.r.t. local air mass (rad)                                                             |

In Eq. (D.7) it is assumed the thrust vector is coincident with the velocity vector. Assume first order responses for the thrust, lift and bank angle as follows:

$$\begin{aligned}\dot{T} &= -p_T T + p_T T_c \\ \dot{L} &= -p_L L + p_L L_c \\ \dot{\phi} &= -p_\phi \phi + p_\phi \phi_c\end{aligned}\tag{D.8}$$

Where  $T_c$ ,  $L_c$ ,  $\phi_c$  are the commanded thrust, lift and bank angle, respectively, and  $p_T$ ,  $p_L$ ,  $p_\phi$  are constants that define the speeds of response for the thrust, lift and bank angle. Note that these quantities are limited, which are defined as

$$\begin{aligned}
0 &\leq T \leq T_{\max} \\
L &\leq K_{L_{\max}} V^2 \\
-\phi_{\max} &\leq \phi \leq +\phi_{\max}
\end{aligned} \tag{D.9}$$

We do not model any release of stores, etc. and as such the aircraft mass rate model is given by

$$\dot{M} = \dot{m}_f = -K_f T \tag{D.10}$$

where  $\dot{m}_f$  is the fuel flow rate, assumed to be proportional to the thrust. Hence,

$$M(t) = M_o + \int_{t_o}^t \dot{M}(t) dt = M_o - K_f \int_{t_o}^t T(t) dt \tag{D.11}$$

where  $M_o$  is the initial mass, and the gross takeoff weight is  $W_{TO} = M_o g$ .

### Control Laws

For general flight, the commanded velocity, flight path and heading angles should be followed. This can be achieved by the following control laws governing the commanded thrust, lift and bank angle, given by

$$\begin{aligned}
T_c &= K_{T2} x_T + K_{T1} \dot{x}_T \\
\dot{x}_T &= M(V_c - V) \\
L_c &= K_{L2} x_L + K_{L1} \dot{x}_L \\
\dot{x}_L &= M V_c (\gamma_c - \gamma) \\
\gamma_c &= K_{gp} (H_c - H_i) \\
\phi_c &= K_{\phi 1} \frac{V_c}{g} (\psi_c - \psi) + K_{ct} F_{ct}
\end{aligned} \tag{D.12}$$

where  $V_c$ ,  $\psi_c$  are the commanded airspeed, and bearing angle, respectively. And, for the flight leg between waypoints  $\{wp_x(j), wp_y(j), wp_h(j)\}$  and  $\{wp_x(j+1), wp_y(j+1), wp_h(j+1)\}$ ,

$$H_c = wp_h(j+1) - \tan(\gamma_{i_c}) F_{ig}. \tag{D.13}$$

Where  $\gamma_{i_c}$  is the vertical angle between the waypoints and  $F_{ig}$  is the distance “to go” until arriving at the next waypoint. The distance to go is determined by:

$$p_1 = (X_i - wp_x(j))\hat{i} + (Y_i - wp_y(j))\hat{j} + 0\hat{k} \tag{D.14}$$

$$p_2 = (X_i - wp_x(j+1))\hat{i} + (Y_i - wp_y(j+1))\hat{j} + 0\hat{k} \tag{D.15}$$

$$\lambda = \frac{P_2 - P_1}{\|P_2 - P_1\|} \tag{D.16}$$

$$\lambda_p = \hat{k} \times \lambda \quad (D.17)$$

$$F_{tg} = \|\lambda_p \times p_2\| \quad (D.18)$$

Similarly, for the cross-track error,  $F_{ct}$ , in Equation (D.12),

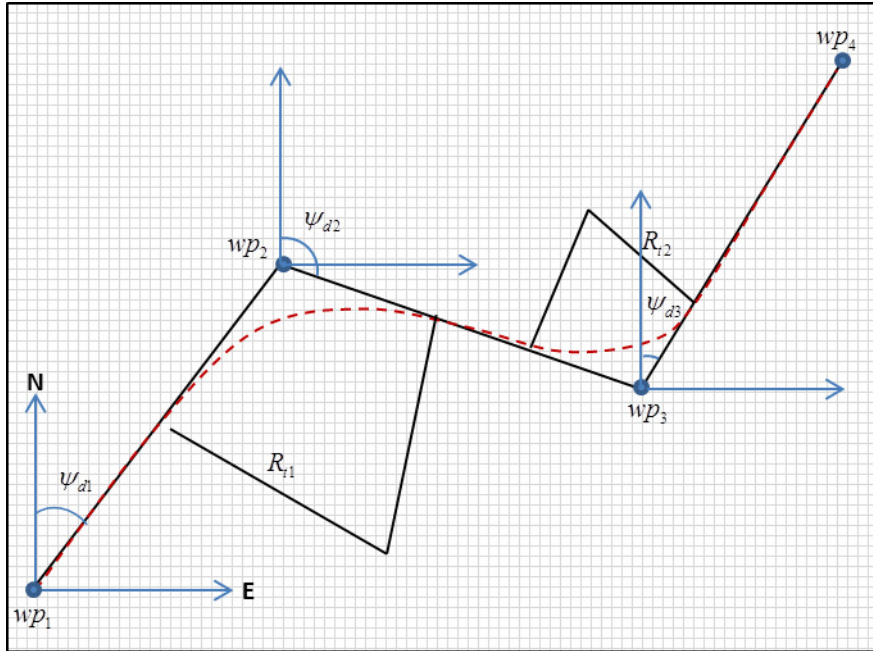
$$F_{ct} = \|\lambda \times p_1\|. \quad (D.19)$$

Also,  $\psi_c$  in Eq. (D.12), is the bearing between the two waypoints. Note that care should be taken to ensure that no discontinuities arise in the  $(\psi_c - \psi)$  difference. In the current effort, continuous wrapping of the individual quantities ( $\psi_c$  and  $\psi$ ) is used to protect against this discontinuity.

More complex trajectory tracking schemes could be augmented to the control laws to follow curved paths, for example. However, straight line, way point to way point guidance should serve our purposes in this project.

### Fly-By Way Point Following

Consider the fly-by waypoint following geometry shown in Figure D.5. Here, the objective is not to have the vehicle fly directly over the waypoints, but rather use the waypoints to define desired headings for each leg of the mission. Four waypoints with their three associated legs and three desired headings,  $\psi_{d1}$ ,  $\psi_{d2}$ ,  $\psi_{d3}$ , are shown in the figure.



**Figure D.5. Waypoint Following Geometry**

The turn radius for each turn can be approximated by

$$R_t \approx \frac{V_d^2}{g \tan(|\phi_{avg}|)} \quad (D.20)$$

where the “average” bank angle is approximated by

$$|\phi_{avg}| = \min \left\{ \left| K_{\phi 1} \frac{V_d}{g} \psi_{avg} \right|, \phi_{max} \right\} \quad (D.21)$$

and, for the first turn,

$$\psi_{avg} = \left( \frac{\psi_{d2} - \psi_{d1}}{2} \right) \quad (D.22)$$

(similarly for the second turn). Command for the next desired heading should be initiated a distance  $D_{turn}$  before the approaching waypoint, where

$$D_{turn} = R_t |\tan(\psi_{avg})| \quad (D.23)$$

A summary of the equations needed to simulate a UAS’s motion is given in Table D.5.

**Table D.5. Summary of Equations for UAS Simulation**

**Inputs:**

- (1) Waypoint locations:  $\{wp_1, \dots, wp_n\} = \{(X_{i1}, Y_{i1}, H_{i1}), \dots, (X_{in}, Y_{in}, H_{in})\}$
- (2) Desired velocities between each waypoint:  $\{V_{c1}, \dots, V_{cn}\}$
- (3) Steady winds:  $\{w_x, w_y, w_h\}$

**Geometry:**

- (1)  $\{\gamma_{ic}\}$  for each leg, (2)  $\{\psi_c\}$  for each leg

**Control Laws:**

|                                                                     |                                                                        |
|---------------------------------------------------------------------|------------------------------------------------------------------------|
| $T_c = K_{T2} \dot{x}_T + K_{T1} \dot{x}_T$                         | $p_1 = (X_i - wp_x(j))\hat{i} + (Y_i - wp_y(j))\hat{j} + 0\hat{k}$     |
| $\dot{x}_T = M(V_c - V)$                                            | $p_2 = (X_i - wp_x(j+1))\hat{i} + (Y_i - wp_y(j+1))\hat{j} + 0\hat{k}$ |
| $L_c = K_{L2} \dot{x}_L + K_{L1} \dot{x}_L$                         | $\lambda = \frac{P_2 - P_1}{\ P_2 - P_1\ }$                            |
| $\dot{x}_L = M V_c (\gamma_c - \gamma)$                             | $\lambda_p = \hat{k} \times \lambda$                                   |
| $\gamma_c = K_{gp} (H_c - H_i)$                                     | $F_{tg} = \ \lambda_p \times p_2\ $                                    |
| $H_c = wp_h(j+1) - \tan(\gamma_{ic}) F_{tg}$                        | $F_{ct} = \ \lambda \times p_1\ $                                      |
| $\phi_c = K_{\phi 1} \frac{V_c}{g} (\psi_c - \psi) + K_{ct} F_{ct}$ |                                                                        |

### **Aircraft Governing Equations of Motion**

$$\begin{aligned}\dot{T} &= -p_T T + p_T T_c & \text{I.C. } T(0) = T_o, \quad 0 \leq T \leq T_{\max} \\ \dot{L} &= -p_L L + p_L L_c & \text{I.C. } L(0) = L_o, \quad 0 \leq L \leq K_{L_{\max}} V^2 \\ \dot{\phi} &= -p_\phi \phi + p_\phi \phi_c & \text{I.C. } \phi(0) = \phi_o, \quad -\phi_{\max} \leq \phi \leq +\phi_{\max} \\ \dot{M} &= -K_f T, \quad \text{I.C. } M(0) = M_o\end{aligned}$$

$$\begin{aligned}\dot{V} &= \frac{T - D}{M} - g \sin(\gamma), \quad D = C_{D1} V^2 + C_{D2} \frac{L^2}{V^2}, \quad \text{I.C. } V(0) = V_o \\ \dot{\gamma} &= \frac{L}{MV} \cos(\phi) - \frac{g}{V} \cos(\gamma), & \text{I.C. } \gamma(0) = \gamma_o \\ \dot{\psi} &= \frac{L}{MV} \frac{\sin(\phi)}{\cos(\gamma)}, & \text{I.C. } \psi(0) = \psi_o\end{aligned}$$

### **Calculation of Inertial States (Truth Values for Plotting)**

$$\begin{aligned}\dot{X}_i &= V_i \cos(\gamma_i) \cos(\psi_i) & V_i &= \sqrt{\dot{X}_i^2 + \dot{Y}_i^2 + \dot{H}_i^2} \\ \dot{Y}_i &= V_i \cos(\gamma_i) \sin(\psi_i) & \gamma_i &= \sin^{-1} \left( \frac{\dot{H}_i}{V_i} \right) = \tan^{-1} \left( \frac{\dot{H}_i}{\sqrt{\dot{X}_i^2 + \dot{Y}_i^2}} \right) \\ \dot{H}_i &= V_i \sin(\gamma_i) & \psi_i &= \sin^{-1} \left( \frac{\dot{Y}_i}{V_i \cos(\gamma_i)} \right) = \tan^{-1} \left( \frac{\dot{Y}_i}{\dot{X}_i} \right)\end{aligned}$$

## **D.4 Breguet Range Equation**

We present the Breguet range equation in this chapter as it involves basic aircraft performance equations. The reason for investigating the range equation is that a mission management level RTA system may check that there will always be enough fuel remaining for the UAS to safely fly back to base. This check will be discussed further in the next chapter. We use the classical Breguet range equation [Waitz 2003] as a *notional* means of a certified check. More complicated schemes would be used in a real world development of such an RTA check, but such schemes are in operational use today to calculate required fuel for a particular flight plan, and are therefore certified. For commercial aircraft, determining the optimal fuel weight is critical for cost savings, so each aircraft is loaded with only enough fuel to fly to the intended destination and one defined alternate destination with a certain percentage of extra fuel and range for contingencies. Therefore, current required fuel load calculations may involve complex optimization schemes that take into account factors that affect rate of fuel burn, such as ambient temperature, aircraft speed, and aircraft altitude. The fuel burn rate also depends on airplane weight, which changes as fuel is burned. The mass rate equation given in Eq. D.10 obviously does not include these higher order effects.

The Breguet range equation assumes steady, level flight, in which thrust equals drag and lift equals weight, or

$$T = D, \quad L = W, \quad \text{or} \quad W = D \left( \frac{L}{D} \right) = T \left( \frac{L}{D} \right) \quad (\text{D.24})$$

The weight of the aircraft changes in response to the fuel that is burned, given by (see Eq. (D.11))

$$\frac{dW}{dt} = -\dot{m}_f g \quad (\text{D.25})$$

We assume here the UASs in our challenge problem (discussed in the next chapter) are equipped with gas powered propeller engines, which have a propulsion system efficiency defined by

$$\eta = \frac{TV}{\dot{m}_f e_f} \quad (\text{D.26})$$

where  $e_f$  is defined as the fuel energy per unit mass. The rate of weight reduction due to fuel usage can be written as

$$\frac{dW}{dt} = -\dot{m}_f g = \frac{-W}{\left( \frac{L}{D} \right) \frac{T}{\dot{m}_f g}} = \frac{-WV}{\frac{e_f}{g} \left( \frac{L}{D} \right) \frac{TV}{\dot{m}_f e_f}} = \frac{-WV}{\frac{e_f}{g} \left( \frac{L}{D} \right) \eta} \quad (\text{D.27})$$

Rearranging and integrating gives

$$\frac{dW}{W} = \frac{-Vdt}{\frac{e_f}{g} \left( \frac{L}{D} \right) \eta} \Rightarrow \ln(W) = \frac{-tV}{\frac{e_f}{g} \left( \frac{L}{D} \right) \eta} + \ln(W_o) \quad (\text{D.28})$$

Solving for time,

$$t = -\frac{L}{D} \eta \frac{e_f}{gV} \ln \left( \frac{W}{W_o} \right) \quad (\text{D.29})$$

Now define the desired final weight of the vehicle as its empty weight plus some margin, or

$$W_f = W_{\text{empty}} + W_{\text{margin}} \quad (\text{D.30})$$

If  $W_o$  represents the current weight of the vehicle at any given time in flight, then the remaining flight time is given by

$$t_{\text{remain}} = -\frac{L}{D} \eta \frac{e_f}{gV} \ln \left( \frac{W_f}{W_o} \right) \quad (\text{D.31})$$

Range remaining is estimated to be



$$R_{remain} = V \cdot t_{remain} \quad (D.32)$$

Eqs. (D.31) and (D.32) give us the Breguet range equation for propeller driven aircraft,

$$R_{remain} = -\frac{L}{D} \eta \frac{e_f}{g} \ln \left( \frac{W_f}{W_o} \right) \quad (D.33)$$

Since

$$L = \frac{1}{2} \rho V^2 S C_L, \quad D = \frac{1}{2} \rho V^2 S C_D \quad (D.34)$$

where  $\rho$  is the air density and  $S$  is the wing planform area, we may also write the range equation in terms of the lift and drag coefficients as

$$R_{remain} = -\frac{C_L}{C_D} \eta \frac{e_f}{g} \ln \left( \frac{W_f}{W_o} \right) = +\frac{C_L}{C_D} \eta \frac{e_f}{g} \ln \left( \frac{W_o}{W_f} \right) \quad (D.35)$$

Propeller engine efficiency is also often expressed in terms of its specific fuel consumption, SFC, which is defined as the mass flow rate of fuel per unit of thrust. In this case, it can be shown that the range equation is expressed as

$$R_{remain} = \frac{C_L}{C_D} \frac{V}{g \cdot SFC} \ln \left( \frac{W_o}{W_f} \right) \quad (D.36)$$

Eqs. (D.35) and (D.36) represent two basic forms of the range equation. There are many additional, more complex variants that take into account trim conditions and other factors related to the power plant. However, for our demonstration purposes here, the above expressions should suffice. In fact, for low value UASs that have limited on board computing capabilities, such simple range equations may best, given enough margin in defining  $W_f$ .

In summary, the RTA check on whether there is enough fuel remaining to safely get back to base would, at each time step, calculate  $R_{remain}$ , given  $W_o$ , and compare this value to the required range to traverse the current baseline plan back to base.

## D.5 UAS Vehicle Models Used in Challenge Problem Demonstrations

For simulation platforms, we will consider a morphing wing vehicle that is an initial model of NextGen Aeronautics' MFX concept platform, shown in Figure D.6. This vehicle falls within the class of small UAS platforms. The 6-DOF model of this platform is used in the inner-loop RTA challenge problem, and a 3-DOF equivalent of this vehicle is used in the mission management level challenge problem. These challenge problems are presented in the next chapters.

We will also investigate heterogeneous fleets for the mission management challenge problem, and for that reason, we may consider two other types of UASs, denoted UAS\_A and UAS\_B. Let us consider, for example, the vehicle data from the examples for Group 2/Group 3 given in

Table D.2, the ScanEagle and Shadow. These vehicles are shown in Figure D.7 and specifics on their vehicle and performance characteristics are given in Table D.6.



**Figure D.6. Morphing Wing Concept Vehicle**



**Figure D.7. Two UAS Vehicles Considered in Challenge Problem**

**Table D.6. Characteristics of Example UAS Platforms for Challenge Problem**

|                     | UAS_A (ScanEagle)                  | UAS_B (Shadow)             |
|---------------------|------------------------------------|----------------------------|
| Manufacturer        | Boeing/Insitu                      | AAI Corp.                  |
| Length              | 5.6 ft                             | 11.2 ft                    |
| Wingspan            | 10.2 ft                            | 14 ft                      |
| Empty Weight        | 35 lbs                             | 186 lbs                    |
| Fully Loaded Weight | 48.5 lbs                           | 375 lbs                    |
| Powerplant          | 1x2-stroke 3W piston engine, 15 hp | 1xAR741-1101 Wankel, 38 hp |
| Maximum Speed       | 80 knots/92 mph                    | 110 knots/127 mph          |
| Cruise Speed        | 60 knots/69 mph                    | 70 knots/81 mph            |
| Endurance           | 24 hrs                             | 9 hrs                      |
| Operational Range   | 100 miles                          | 68 miles/59 Nm             |
| Service Ceiling     | 19,500 ft                          | 15,000 ft                  |

## **Appendix E**

### **Fault Tree and Failure Modes Analyses for RTA Protected Systems**

#### **E.1 Introduction**

Prior sections of this report have discussed in depth some of the potential failure modes of the Mission Management System. Analysis of this type represents one of the two principle approaches for doing failure analysis, often through the formality of a FMECA. The other principle approach is usually done using Fault Tree Analysis (FTA). The particular RTA architecture that is being proposed for the challenge problem is well-suited for FTA during both the design process and to support the safety certification process. Understanding the differences between the two failure analysis approaches and the applicability of each in an RTA equipped system is critical for ensuring the overall system safety. The following sections will discuss the two principle approaches, present guidelines for doing a formal FTA and then include a short example showing the benefits of FTA during design of an RTA for the challenge problem. The Risk Mitigation approach as shown in the Year 1 report will then be presented as a more detailed example of the use of FTA to make a safety certification argument in regards to specific undesired system behavior, namely vehicle collision.

#### **E.2 Failure Analysis Overview and Applicability to RTA**

Failure Analysis methodologies fall into two broad approaches: inductive methods and deductive methods. FTA is a deductive method, and FMECA is an inductive method. The two methods are fundamentally different, and should not be thought of as being interchangeable. Deductive methods start with an undesired system behavior and deduce the possible causes of the behavior. On the other hand, inductive methods start with a particular failure of a system component and determine the effects on the system behavior.

##### Deductive Methods

- Reasoning from the general to the specific
- Top down: Top level events analyzed to reveal possible causes
- Answers the question “How could an event happen?”
- Determines how a specified state can occur
- Analyzes causes

##### Inductive Methods

- Reasoning from the specific to the general
- Bottom up: Base level events analyzed to consider possible effects
- Answers the question “What would happen if...?”
- Determines all the possible states of a system
- Analyzes effects

FTA is the prototypical deductive methodology for conducting failure analysis. An FTA will start with a top level event which describes an undesirable behavior of the system and will

determine the base events which can lead to the top level event. This is done in a sequential manner by determining the direct causes of the top level event and then the direct causes of those events in return and repeating this procedure as needed until the desired basic causes have been determined. An FTA is limited by the (usually pre-determined) scope and resolution of the analysis. The scope represents the breadth of causes which will be included (e.g. will environmental variations be included) and resolution represents the depth to which the analysis will be conducted (i.e. what are the basic “components” whose failures will be considered fundamental and will not be analyzed in greater detail.) Fundamentally, FTA is a qualitative method, but by assigning event probabilities to the base events, quantitative data for the entire tree can be generated, and thus FTA can be effectively used for quantitative risk analysis. FTA is NOT an exhaustive list of system faults, as it is driven by the specified top level event, and this is one of the fundamental ways in which it differs from a FMECA type approach.

An FTA has a number of uses relating to system safety, many of which can aid in establishing a certification argument for an RTA architected system:

- Understand logical connections between baseline events and component failures that lead to top event
- Prioritize contributing factors leading to top event
- Pro-active tool to prevent top level event (thus can provide valuable information during design of RTA, and specification of RTA scope)
- Monitor performance of the system (systematic updating and re-evaluating of FT based on system changes, aging effects, etc.)
- Minimize and optimize resources (target resources to the most critical contributing factors)
- Assist in design of system (including RTA components)
- Diagnostic tool to identify and correct causes of an occurrence of the top level event

It is important to understand how FTA can be useful for a system which includes an RTA architecture as one of its components. Fundamentally, RTA approaches can work in one of two ways, either monitoring a specific advanced component for failures, or monitoring for undesired states of the system. These two approaches for failure monitoring exactly mirror the deductive and inductive methods of doing failure analysis. Although hybrid approaches are certainly possible, much care would be needed to insure that complete coverage of the RTA was provided as neither FTA nor FMECA type failure analysis will match up one-to-one with a hybrid approach. For the challenge problem, the RTA has been architected to essentially monitor for undesired states, and thus our RTA architecture is fundamentally deductive (top-down), determining whether the causes of a faulty state are the advanced components and whether to revert or not (as opposed to trying to determine whether the outputs of an advanced component reflect a failure of the component). With this type of architecture, direct ties can be made between an FTA for a particular undesirable system behavior and the RTA that is monitoring for that behavior. The FTA can be useful both during the design process, in suggesting what behaviors an RTA should be monitoring, and also when conducting Risk Analysis for the entire system (with the RTA in place), demonstrating the reduced probability of the top level event due to the Risk Management provided by the RTA. It should be noted that this does NOT invalidate

the use of FMECA in RTA setting, nor the failure mode analysis shown in prior sections. In fact, FMECA has one clear advantage over FTA in that it provides some assurance of completeness of the analysis. For this reason, FMECA can suggest top level events that might need a fault tree, or that should be addressed with an RTA or to validate an existing fault tree. Furthermore, FMECA analysis will be critical in certifying the system, as it can demonstrate that all possible component failures have been considered.

A notional pair of examples will help illustrate how FTA can be useful during the design of an RTA. Two cases will be considered, but it should note that neither of this is intended to represent the actual system developed for the challenge problem, but are rather for illustrative purposes on the use of FTA. Fault trees for the actual Mission Management System will be shown later in this section. For both of the example cases, the top level (undesired) system behavior will be that the stall angle of attack is exceeded by the vehicle and in both cases a baseline system exists which prevents stall and an advanced guidance system (with RTA) is being designed.

Case 1: The baseline system prevents stall by limiting the elevator deflections on the vehicle. In this case, a fault tree for vehicle stall for the baseline system would not have any entries in the fault tree for guidance components, as guidance commands cannot cause the vehicle to stall, since stall is prevented through limits on elevator deflection, applied independent of guidance commands. The designer of the RTA for the advanced guidance system would generate a separate fault tree for the advanced system, but would similarly see that no advanced guidance components appear in the “vehicle stall” fault tree, for exactly the same reason. Thus, the RTA designer can conclude that monitoring for vehicle stall is unnecessary for the RTA in this particular case and can focus the RTA effort towards other fault trees representing other undesired system behaviors.

Case 2: The baseline system prevents stall by the guidance algorithms limiting the angle of attack. In this case, a different approach has been taken by the baseline system and stall is prevented by the angle of attack limits within the baseline guidance system. Contrary to Case 1, a fault tree for vehicle stall for the baseline system will show guidance component failures, albeit at a very low probability, as the baseline guidance has been certified as being reliable. Once again, the designer of the RTA will generate a new fault tree for the advanced system, and this fault tree will include advanced guidance component failures. At this stage, the power of FTA during RTA design can be seen as several possibilities exist. First, if the probability of failure of the advanced components *in the particular ways represented on this fault tree* are the same or lower than the baseline components, then an RTA should not be needed for vehicle stall. This does not suggest that the advanced components are certifiable (recalling that FTA is driven by system behaviors, not component reliability), but rather that this particular system behavior (vehicle stall) does not need to be monitored by an RTA. The second possibility is that the probability of failure of the advanced component is higher than the baseline component, or cannot be accurately determined. In this case, the RTA designer can finally see a place that an RTA would be appropriate.

These two cases illustrate that FTA can both suggest where an RTA might be needed, but also can show system behaviors that do not need an RTA at all. Furthermore, FTA can illustrate the potential weaknesses of a system in such a way that an alternative to an RTA for the advanced

components might be suggested. For example, the RTA designer in Case 2 above might note the weakness in having vehicle stall prevented as a guidance function, and might suggest using something like the elevator deflection limiting shown in Case 1, and thus obviating the need for RTA at this point in the system. In other words, an RTA designer armed with an FTA for the system is well equipped to suggest changes to the overall system architecture that might be more effective, and more acceptable from a safety standpoint than simply putting an RTA in place for a particular system behavior. If the RTA designer can successfully remove the advanced components from ALL of the system fault trees, then the RTA requirements shrink to nothing and the advanced components cannot harm the system and would be safe to implement with no RTA at all.

Once an RTA architected system has been designed, the fault trees can be used quantitatively to generate probabilities of undesired system behavior and to suggest modifications so the design as well as to generate metrics for safety certification, and risk mitigation tables. A full set of fault trees for the Mission Management System would be extensive and would include advanced and baseline components. For illustration purposes, two fault trees will be considered, Aircraft Loss of Lift, and Vehicle Collision. The former will illustrate the building of a fault tree from basic principles, and the latter will illustrate the use of a fault tree to generate Risk Mitigation tables. Some formal procedures for generating a fault tree will be described first.

### **E.3 FTA Formal Procedures**

It is not the purpose of this section to present a full description of FTA or how it should be conducted. Much of the information presented here and the approach taken are derived from the standard NASA approach to FTA which has been adopted in a number of different industries, and is fully described in [Stamatelatos 2002]. Instead, an overview of the general procedure is shown here, followed by the examples in the following sections. Some general comments are appropriate first. FTA is a failure focused analysis method (as indeed FMECA is as well.) Analysis can, in theory, be done from a success viewpoint instead, by examining the ways in which a system can be insured to be successful, rather than the ways in which it can fail. In fact, a formal method exists for converting a fault tree into its logical complement, a success tree, showing all the ways in which a top level of event (the negation of the fault tree top level event) is guaranteed to occur. However, failure analysis is almost always a more desirable way to analyze the system due to several factors:

- Easier to agree on what constitutes failure than success
- Success often associated with efficiency, and non-discrete evaluations
- Success space often much larger than failure space
- Failure probabilities low, and the math used for overall probability calculations is easier with numbers near 0, rather than numbers near 1

It should be noted, however, that Assume/Guarantee reasoning is often success driven (guaranteeing what will happen, not what will not happen) and it is often useful to invert the Assume/Guarantee reasoning when using it to help with the construction of fault trees.

The interrelationships among events captured by a fault tree are described by three concepts: failure modes, failure mechanisms and failure effects. For a given system, subsystem or component:

- Failure modes are the various ways in which the element can fail
- Failure mechanisms are the immediate underlying causes of the failure modes
- Failure effects are the immediate upstream effects of the failure mode for higher level systems

Failure modes are one level are failure effects for the next level down and failure mechanisms for the next level up. An example is shown in Table E.1, examining the loss of vehicle due to an elevator high command, with the vehicle being the highest level system and the elevator being the lowest level component.

**Table E.1. Failure Mechanism, Mode and Effect Example**

| <b>Description of Event</b> | <b>Vehicle<br/>(system)</b> | <b>Airframe<br/>(subsystem)</b> | <b>Wing<br/>(subsystem)</b> | <b>Elevator<br/>(component)</b> |
|-----------------------------|-----------------------------|---------------------------------|-----------------------------|---------------------------------|
| Loss of Vehicle             | Mode                        | Effect                          |                             |                                 |
| Loss of Lift                | Mechanism                   | Mode                            | Effect                      |                                 |
| Stall angle of attack       |                             | Mechanism                       | Mode                        | Effect                          |
| Elevator high command       |                             |                                 | Mechanism                   | Mode                            |

The main steps for actually performing FTA are as follows:

1. Identify the objective
2. Define the top event
3. Define the scope
4. Define the resolution
5. Define ground rules (procedure, nomenclature)
6. Construct the FT
7. Evaluate the FT
8. Interpret and present the results

Steps 3-5 are usually done in parallel, and can be iterated on as a result of Steps 6 and 7. In particular, the scope and resolution are often redefined as the structure of the fault tree is developed. For relatively small fault trees such as will be shown here, the nomenclature is not as critical as it is not difficult to examine the tree for duplication and other problems, but for a full system FTA, it is important to have well defined procedures and nomenclature. The key principle for successfully constructing the fault tree in Step 6 is to take small steps at each level of the tree, insuring that each level appropriately maps the actual mechanisms by which higher level failure modes are achieved.

#### **E.4 Example of FTA and RTA during System Design – Aircraft Loss of Lift**

Two examples of a fault tree for the Mission Management System will be shown. Recall that fault trees are associated with a single top level undesired system behavior. The undesired system behaviors examined in the two examples will be *Vehicle Loss of Lift* and *Vehicle Collision with Another Vehicle*. The first example, will be used primarily to illustrate the development of a fault tree for system design purposes, illustrating the basic principles described



above. As such, no attempt will be made to address event probabilities or to do risk mitigation analysis. The second example is the Vehicle Collision fault tree, illustrating the use of a fault tree for developing Risk Mitigation tables for the proposed system design.

Following the steps laid out above, a basic fault tree will be shown for the top level event *Vehicle Loss of Lift*. The top level system for this fault tree will be the airframe of the UAS. Explicitly illustrating the steps followed:

Step 1: Identify the objective. The objective of this FTA is to prevent catastrophic loss of vehicle lift capability during flight. Loss of lift during take-off or landing is not part of the objective, nor is unexpectedly low lift that does not lead to vehicle stall and thus vehicle loss.

Step 2: Define the top event. The top level event will be “Loss of Lift on the Vehicle Wing Surface”. Note that the wing is the primary lift surface for the vehicle, representing substantially more lift capability than the body and tails, and thus is appropriate for meeting the objective. This event does not address loss of lift on the tail surfaces, and thus does not encompass loss of control of the vehicle, which would be a separate top level event with its own fault tree.

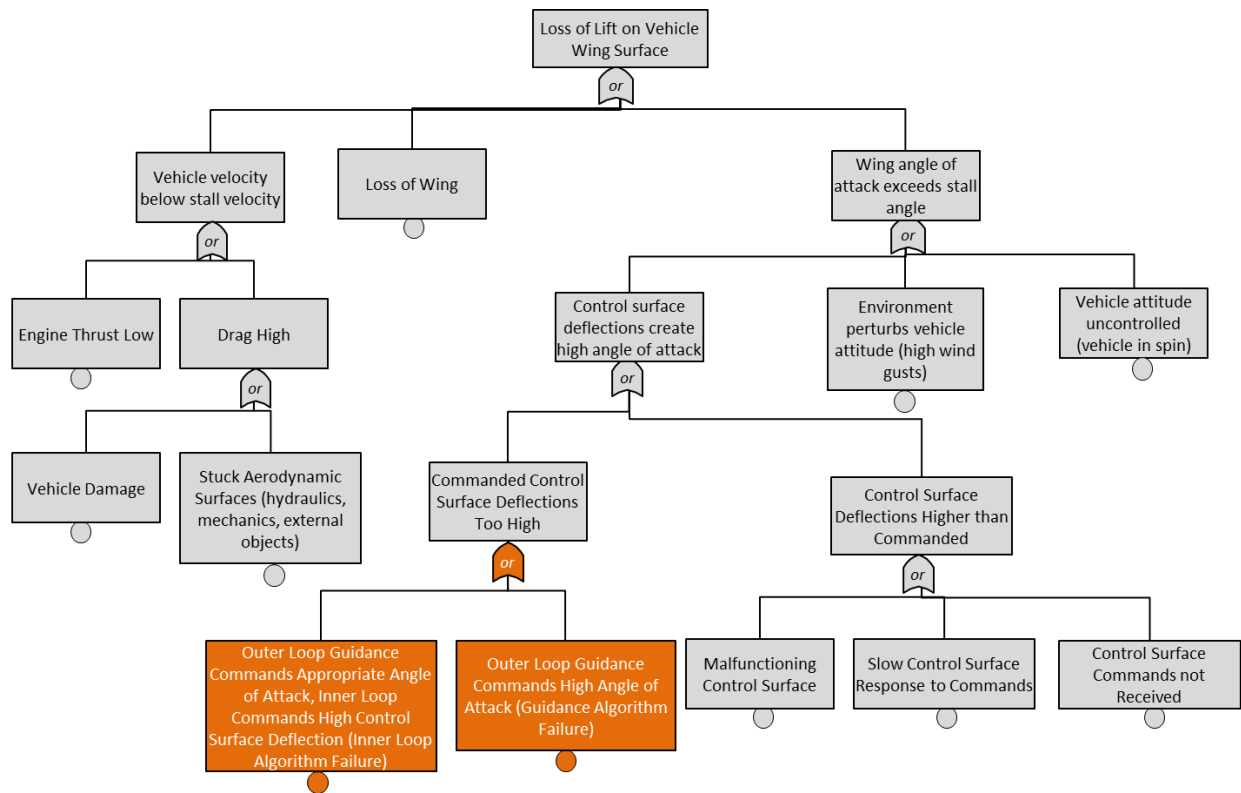
Step 3: Define the scope. For the purposes of this analysis the scope will be limited to environmental factors, engine performance, control surface performance, and guidance and control commands. The scope will not include interactions with other vehicle systems such as health management, and will not include any human interactions.

Step 4: Define the resolution. The resolution of the analysis will vary somewhat depending on the specific system components. Environmental effects will be considered to be baseline events, as will any engine performance issues. System components such as electrical and hydraulic systems and command and control structures will all be considered to be below the resolution of this analysis, and control surface components will be deemed to include all such subcomponents. Guidance and control algorithms will be considered baseline components and further resolution of specific algorithm failure modes will be below the desired resolution.

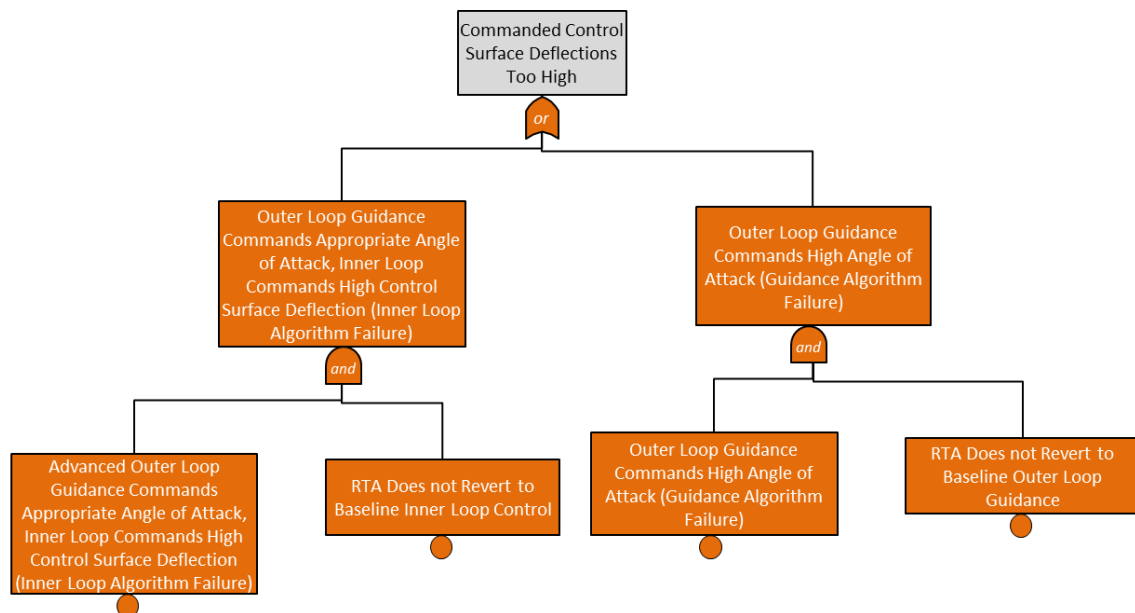
Step 5: Define ground rules. Due to the small size and limited resolution of this fault tree, explicit ground rules will not be defined.

Step 6: Construct the FT. Using the scope and resolution defined above, the following fault tree was generated, one level at a time:

The orange blocks in the diagram represent areas that would be appropriate for an RTA. The fault tree shown assumes no RTA components and could represent either a baseline or an advanced system without an RTA. If the guidance and control components are certified, baseline components then the failure probabilities that would later be assigned to this fault tree would be low. However, for advanced components, the probabilities might be high, or unknown. Thus, an RTA is warranted if another risk mitigation is not available. Thus, a modified fault tree would be generated with RTA components. At the low level of resolution used in this example, this would mean a fairly simple change to the area of the fault tree highlighted in orange in Figure E.1, as shown in Figure E.2.



**Figure E.1. Fault Tree for Loss of Lift on Vehicle Wing Surface**



**Figure E.2. Modified Fault Tree with RTA Components**

## E.5 Example of FTA and RTA during System Design – Ownship Collision

The second example of FTA to be shown is for the top level event *Ownship collides with another vehicle*. Here, we assume collision avoidance functionality resides within the FMS. This is an

alternative framework that provides a richer exploration of the FTA method. This example is designed to show how risk mitigation tables and strategies can be formulated based on the information in a fault tree, and how probabilistic assessments are applied. Following the strict guidelines for generating a fault tree presented in the prior section to the level of resolution desired for this example would result in a very large fault tree that could not be easily presented in a single diagram. Since it is desirable to demonstrate some of the pathways that lead down to basic, component level failures, some liberties have been taken in the usual rigorouslyness that would need to be applied to a full fault tree, leading to the more manageable fault tree presented in the following section and the associated Risk Mitigation tables. As Risk Mitigation includes assessments of relative risk levels, and probability of event occurrence, some standards for probability and criticality that will be applied to this analysis will be presented first.

### E.5.1 Probability and Criticality Levels

In this study, we will present notional probabilities of event occurrences, similar to that shown in Table E.2. which comes from MIL-STD-882. This table shows five categories of “probability levels.” Based on this, we present a probability model shown in Table E.3 which expands the probability definition to eight levels. Note that an actual FTA procedure would develop probability models with numerical values. However, here, for demonstration purposes, we will categorize probability values as shown in the table.

**Table E.2. Failure Probability Levels (MIL–STD–882)**

| Level      | Description                                                                                                |
|------------|------------------------------------------------------------------------------------------------------------|
| Frequent   | Likely to occur in the life of the item or continuously experienced                                        |
| Probable   | Will occur several times in the life of an item or will occur frequently                                   |
| Occasional | Likely to occur sometime in the life of an item or will occur several times                                |
| Remote     | Unlikely but possible to occur in the life of an item or unlikely, but can reasonably be expected to occur |
| Improbable | So unlikely, it can be assumed occurrence may not be experienced. Unlikely to occur, but possible.         |

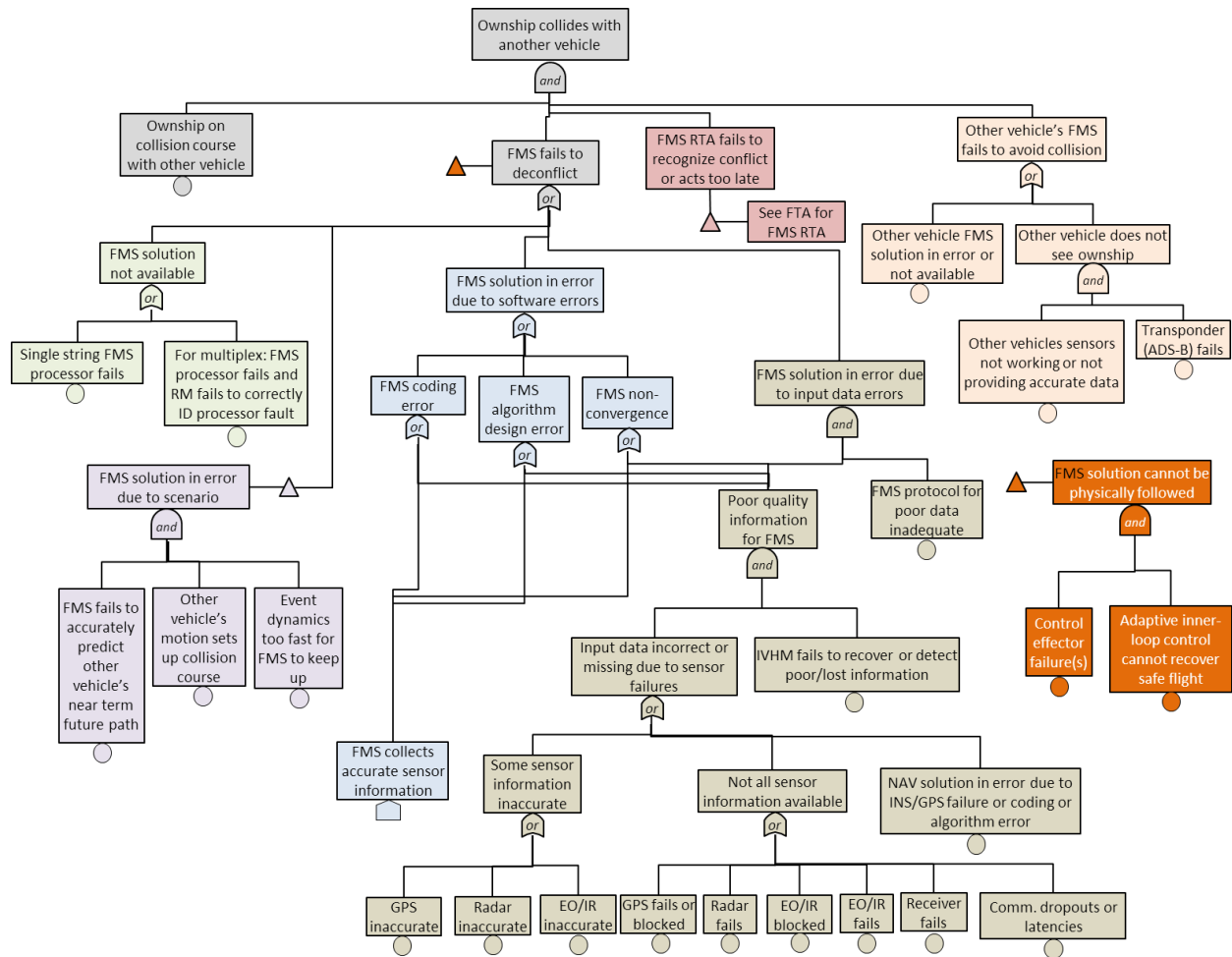
**Table E.3. Notional Probability Categories**

| <b>Label</b> | <b>Probability Definition</b> | <b>Description</b>                                          |
|--------------|-------------------------------|-------------------------------------------------------------|
| <b>EH</b>    | Extremely High                | 0.999999 or greater: Event will almost always occur         |
| <b>VH</b>    | Very High                     | In range of 0.99: Event has a very high chance of occurring |
| <b>H</b>     | High                          | In range of 0.8: Event has a high chance of occurring       |
| <b>LH</b>    | Low High                      | In range of 0.7: Event has a good chance of occurring       |
| <b>M</b>     | Mid-Range                     | In range of 0.3 to 0.7: Event may or may not happen         |
| <b>L</b>     | Low                           | In range of 0.1 to 0.3: Event has a low chance of occurring |
| <b>VL</b>    | Very Low                      | In range of 0.01: Event has a very low chance of occurring  |
| <b>EL</b>    | Extremely Low                 | 0.000001 or lower: Event will almost never occur            |

In Table E.4 we repeat the criticality definitions from DO-178C and will use this model in the FTA study.

**Table E.4. Criticality Level Table (DO-178C)**

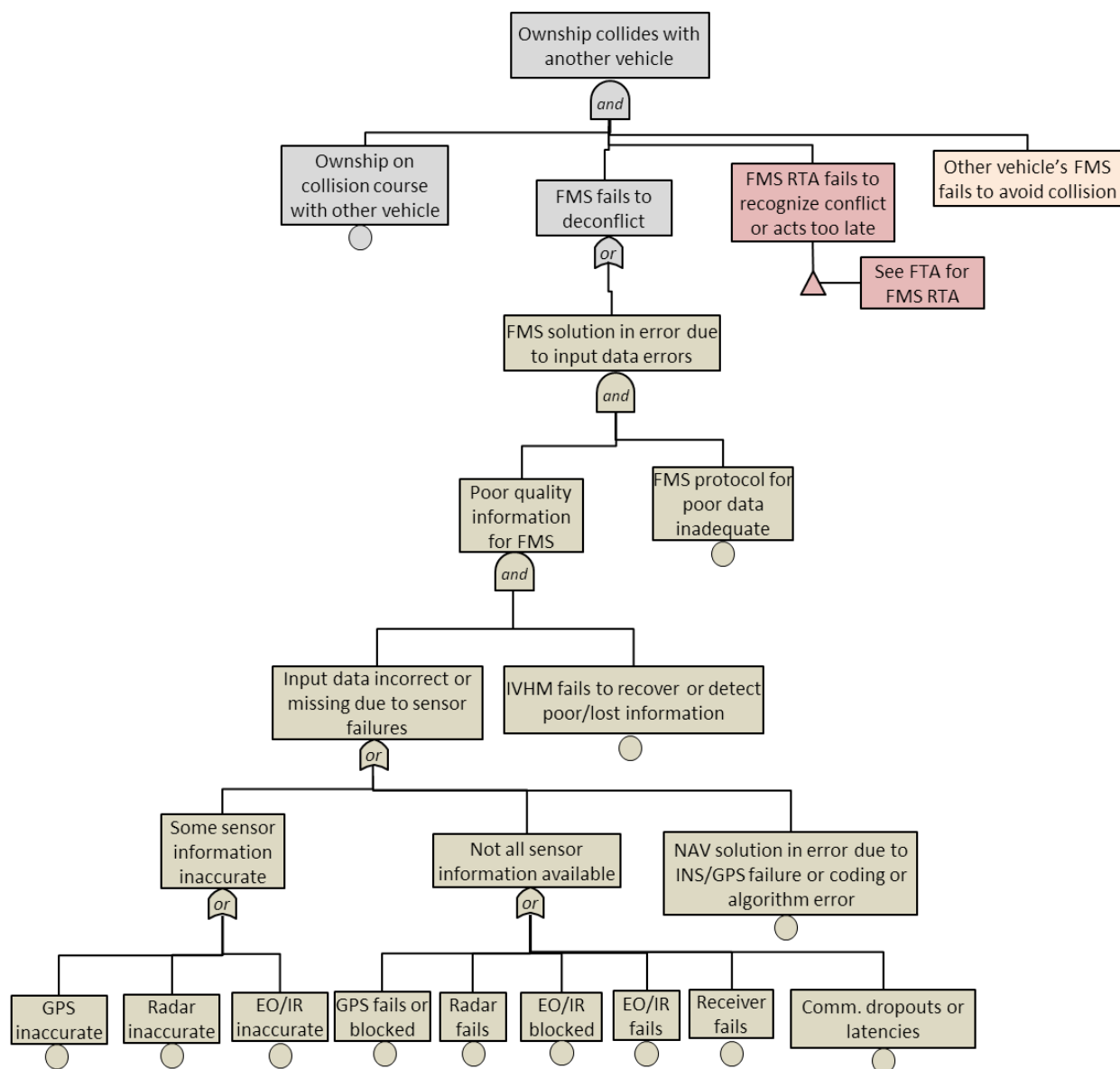
| <b>Level</b> | <b>Failure Condition</b> | <b>Description</b>                                                                                                                                                                                                                                     |
|--------------|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>A</b>     | Catastrophic             | Failure may cause a crash. Error or loss of critical function required to safely fly and land aircraft.                                                                                                                                                |
| <b>B</b>     | Hazardous                | Failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the aircraft due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers. (Safety-significant) |
| <b>C</b>     | Major                    | Failure is significant, but has a lesser impact than a Hazardous failure (for example, leads to passenger discomfort rather than injuries) or significantly increases crew workload. (safety related)                                                  |
| <b>D</b>     | Minor                    | Failure is noticeable, but has a lesser impact than a Major failure (for example, causing passenger inconvenience or a routine flight plan change)                                                                                                     |
| <b>E</b>     | No Effect                | Failure has no impact on safety, aircraft operation, or crew workload.                                                                                                                                                                                 |



**Figure E.3. FMS FTA for Collision Avoidance Function with Single Vehicle Conflict**

### E.5.2 Fault Tree Analysis due to Data Input Errors

The FTA diagram shown in Figure E.4 isolates the fault pathway for data input error occurrences.



**Figure E.4. Fault Tree Analysis for Collision Avoidance Function due to Data Input Errors**

The following table gives a description of the risk mitigation processes required to bring the probability of the ownship colliding with another vehicle to an acceptable value for successful certification. Here,  $P_o$  = probability of occurrence.

**Table E.5. Risk Mitigation Processes for Data Input Errors**

| <b>Event</b>                                        | <b>Risk Mitigation Process/Description</b>                                                                                                                                                                                                                                                                                                                                                                       | <b>P<sub>o</sub> w/o Risk Mitigation</b> | <b>P<sub>o</sub> with Risk Mitigation</b> | <b>Criticality</b>                                |
|-----------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------|-------------------------------------------|---------------------------------------------------|
| GPS, Radar, EO/IR, other sensors inaccurate or fail | Use high quality hardware; maintenance should follow recommended/certified practices and schedule; use redundant sensors and multiplex architectures. Use analytic redundancy algorithms to recover information using other sensors. Criticality ranges: 1 if common mode failure eliminates critical, required information (but probability of occurrence = VL); 4-5 if loss of data can be estimated/recovered | L                                        | VL                                        | A - E                                             |
| GPS, EO/IR temporarily blocked                      | Use in coordination with IMU; Develop algorithms that are robust to data dropouts; Probability is high – common occurrence Criticality can be = 1 during tight formation or avoiding intruders                                                                                                                                                                                                                   | H                                        | H                                         | A-C                                               |
| Receiver fails                                      | Use high quality radios; maintenance should follow recommended/certified practices and schedule; use redundant radios and multiplex architectures.                                                                                                                                                                                                                                                               | VL                                       | EL                                        | A-C                                               |
| Comm. dropouts or latencies                         | Use high quality radios; maintenance should follow recommended/certified practices and schedule; use redundant radios and multiplex architectures. This is a common occurrence and systems should be constructed to be robust to this.                                                                                                                                                                           | H                                        | M                                         | D-E                                               |
| NAV solution in error                               | Use high quality hardware; maintenance should follow recommended/certified practices and schedule; NAV systems typically certified – probability of algorithm/coding error = EL; Criticality = 1 if permanent NAV failure, otherwise 3-5 if errors temporary/solution can be recovered                                                                                                                           | VL                                       | EL                                        | A-E                                               |
| Summary from above: input data incorrect or missing | Probability is high for temporary drop outs, very low for permanent loss of critical information; industry standards exist for risk mitigation of sensor failures                                                                                                                                                                                                                                                | H                                        | H                                         | A, P <sub>o</sub> = VL<br>C-E, P <sub>o</sub> = L |

**Table Continued**

| <b>Event</b>                                                       | <b>Risk Mitigation Process/Description</b>                                                                                                                                                                                                                                                                         | <b>P<sub>o</sub> w/o Risk Mitigation</b> | <b>P<sub>o</sub> with Risk Mitigation</b> | <b>Criticality</b>      |
|--------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------|-------------------------------------------|-------------------------|
| IVHM fails to recover or detect poor/lost information              | FDIR algorithms and IVHM systems currently under wide development and use. Risk mitigation is to accurately develop and certify FDIR algorithms for each specific application; P <sub>o</sub> = EL; criticality can be = 1 if critical, required information lost or in error                                      | VL                                       | EL                                        | A-B                     |
| Summary from above: poor quality information for FMS               | Develop FMS to be robust to input data errors and temporary drop outs. Since sensor errors/drop outs common, this must be addressed. Criticality can be = 1 if critical, required information permanently lost or in error (not possible to be robust to this). For temporary errors, criticality = 4-5            | H                                        | H                                         | A-E                     |
| FMS protocol for poor data inadequate                              | Develop FMS to be robust to input data errors and temporary drop outs. Perform design time V&V as much as possible with input error dispersion studies. Develop RTA checks for this occurrence (assuming no common mode failures for both FMS and RTA – i.e. input data errors cannot cause failure of RTA system) | VL                                       | EL                                        | A-E                     |
| Summary from above: FMS solution in error due to input data errors | Develop RTA checks for this occurrence. RTA will not change probability of this occurrence, but can change its criticality. (assuming no common mode failures for both FMS and RTA – i.e. input data errors cannot cause failure of RTA system)                                                                    | VL                                       | VL                                        | A w/o RTA<br>D-E w/ RTA |
| FMS fails to deconflict                                            | Develop FMS to be robust to input data errors and temporary drop outs during deconfliction protocols. Develop RTA checks for this occurrence (assuming no common mode failures for both FMS and RTA – i.e. input data errors cannot cause failure of RTA system)                                                   | VL                                       | VL                                        | A w/o RTA<br>D-E w/ RTA |
| Ownship on collision course with other vehicle                     | Keep vehicles separated by enough distance so each vehicle has time to deconflict – minimum separation should increase with increasing winds/turbulence and other adverse environmental factors.                                                                                                                   | VL                                       | VL                                        | A-E                     |



**Table Continued**

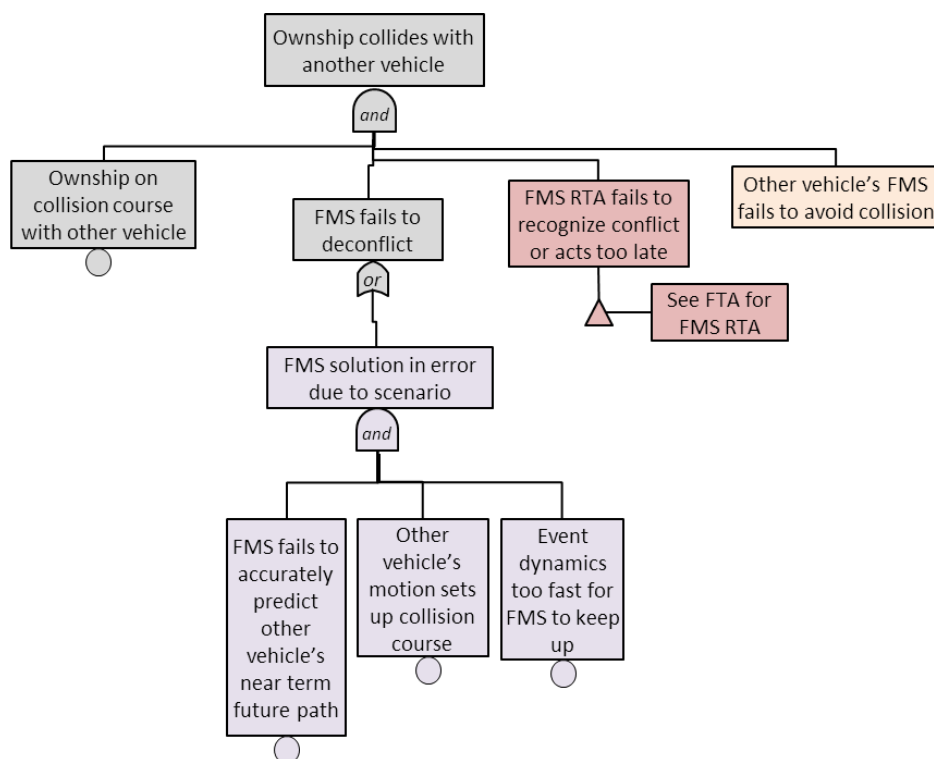
| <b>Event</b>                                              | <b>Risk Mitigation Process/Description</b>                                                                                                                                                                                                                                                                                                                                                                              | <b>P<sub>o</sub> w/o Risk Mitigation</b> | <b>P<sub>o</sub> with Risk Mitigation</b> | <b>Criticality</b> |
|-----------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------|-------------------------------------------|--------------------|
| FMS RTA fails to recognize conflict or acts too late      | Develop RTA to be accurate in its critical collision avoidance functionality.<br>Develop RTA to be more robust to input data errors than FMS.                                                                                                                                                                                                                                                                           | VL                                       | EL                                        | A                  |
| Other vehicle's FMS fails to avoid collision              | Risk mitigation is same as above.<br>Probability of occurrence = EL unless common mode failure (FMSs on both ships in error due to same reason – e.g. both have flown into fog/cloud cover or smoke and some algorithm error results in total system failure due to loss of EO/IR data). Other risk mitigation – do not allow operations in adverse environmental conditions that critically effect sensor performance. | EL                                       | EL                                        | A                  |
| Summary from above: Ownship collides with another vehicle | Risk mitigation is same as above.<br>Probability of occurrence << EL as long as proper minimum separation protocols followed.<br>Ultimately, here is where we must argue that risk is mitigated to certifiable levels with a working RTA. Must prove that the certified RTA will correctly arrest any ensuing collision course.                                                                                         | EL                                       | <<EL                                      | A                  |

In summary, sensor failures and sensor inaccuracies are common occurrences and risk mitigation is largely a process of making all subsystems robust to such failures/errors and employing channel redundancies. Sensor system architectures will depend on criticality of equipment, mission roles and required safety. Low value UASs flown only in combat may have single string systems, whereas high value assets or vehicles flown over populated areas may require triplex systems, redundancy management and high quality sensors. RTA can play a role in checking certain aspects of the FMS's performance if sensor failures/inaccuracies are identified and quantified. Key items for certification are:

1. IVHM and FDIR systems are required to perform to specified levels so that probability of long term critical loss of information = EL.
2. RTA system is required to be robust to sensor failures/inaccuracies to specified levels.
3. Proper protocols should be in place for identifying when the RTA system and the FMS can no longer be trusted under current sensor failures (e.g. activate baseline procedures and fly back to base, or transmit distress signal and have remote human operator take control of vehicle).

### **E.5.3 Fault Tree Analysis due to Adverse Scenarios**

The FTA diagram shown in Figure E.5 isolates the fault pathway for unforeseen adverse scenario occurrences.



**Figure E.5. Fault Tree Analysis for Collision Avoidance Function due to Scenario**

The following table gives a description of the risk mitigation processes required to bring the probability of the ownship colliding with another vehicle to an acceptable value for successful certification.

**Table E.6. Risk Mitigation Processes for Adverse Scenarios**

| Event                                                                 | Risk Mitigation Process/Description                                                                                                                                                                                                                                                                               | P <sub>0</sub> w/o Risk Mitigation | P <sub>0</sub> with Risk Mitigation | Criticality |
|-----------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------|-------------------------------------|-------------|
| FMS fails to accurately predict other vehicle's near term future path | Require separation distances and update rates to minimize effect of prediction errors. Require operation only in winds/turbulence conditions that do not cause erratic vehicle attitude changes (conditions that can be handled by vehicle). Criticality level depends on velocity and distance between vehicles. | M                                  | L to VL                             | A - E       |
| Other vehicle's motion sets up collision course                       | Risk mitigation for all FMS applies. (See other tables – FMS solution should never set up collision course. Criticality level depends on velocity and distance between vehicles.                                                                                                                                  | L to VL                            | L to VL                             | A-C         |

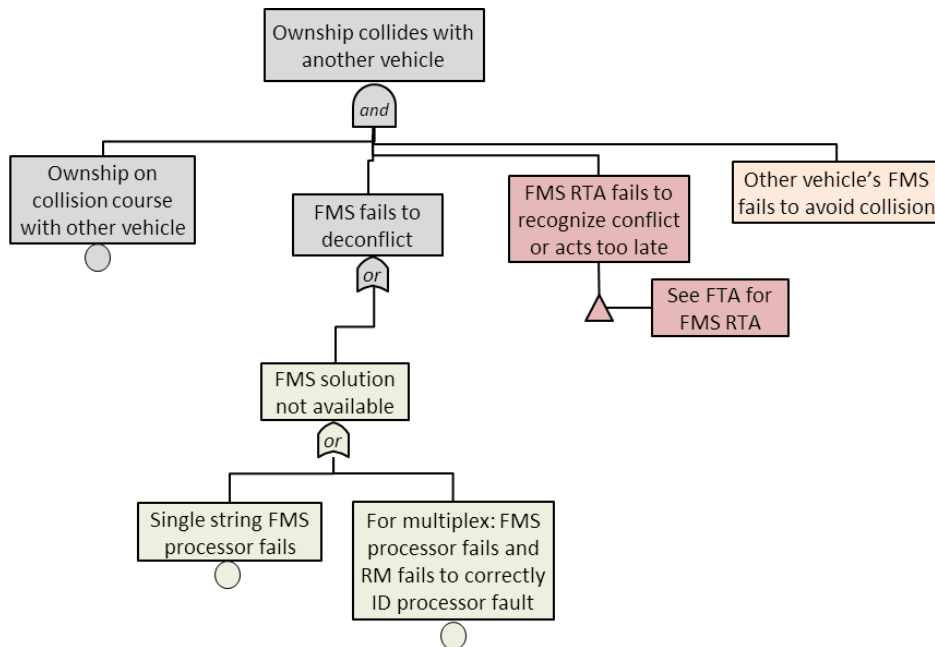
**Table Continued**

| Event                                                                             | Risk Mitigation Process/Description                                                                                                                                                                               | P <sub>o</sub> w/o Risk Mitigation | P <sub>o</sub> with Risk Mitigation | Criticality |
|-----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------|-------------------------------------|-------------|
| Event dynamics too fast for FMS to keep up                                        | Require ratings on separation distances, velocities and update rates such that FMS can provide correct solution in the given amount of time. Criticality level depends on velocity and distance between vehicles. | VL                                 | EL                                  | A-C         |
| Summary from above: FMS solution in error due to scenario and fails to deconflict | Develop RTA checks for this occurrence: e.g. compare current rate of dynamics versus some predetermined “never to exceed” bandwidth.                                                                              | VL                                 | VL to EL                            | A-C         |

See the previous table for descriptions of the higher levels of the FTA diagram shown in Figure E.4.

#### E.5.4 Fault Tree Analysis due to Processor Failure

The FTA diagram shown in Figure E.6 isolates the fault pathway for processor failures. Following the figure, Table E.7 gives a description of the risk mitigation processes required to bring the probability of the ownship colliding with another vehicle to an acceptable value for successful certification.



**Figure E.6. Fault Tree Analysis for Collision Avoidance Function due to Processor Failure**

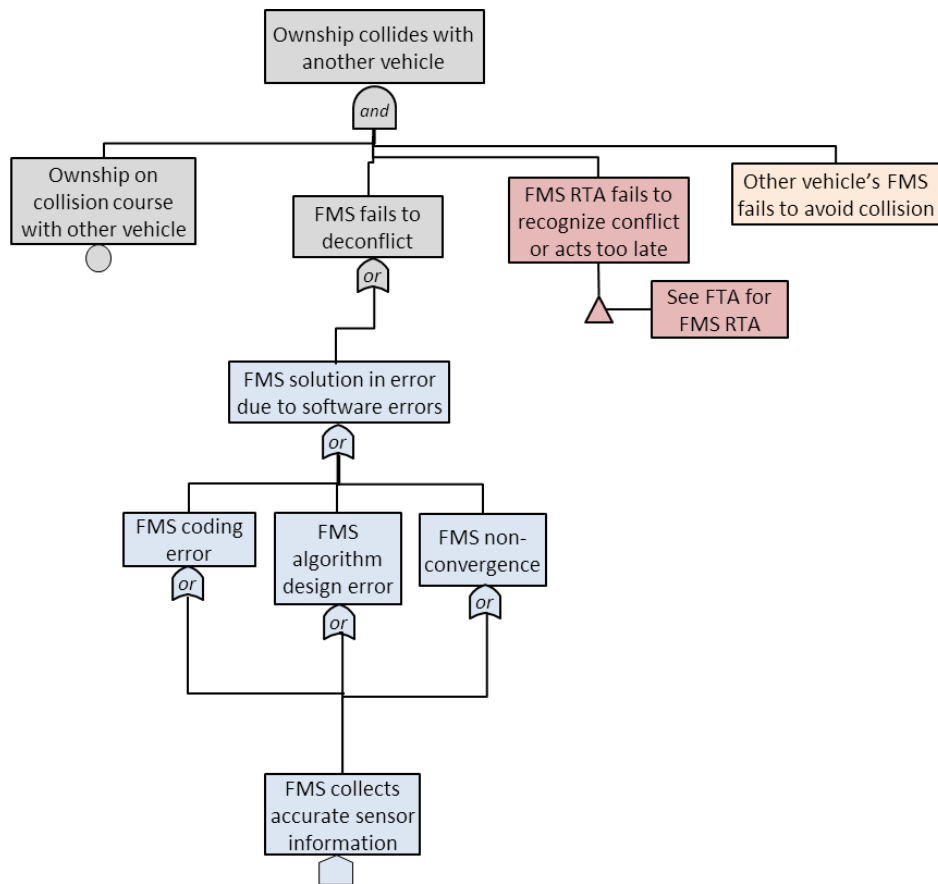
**Table E.7. Risk Mitigation Processes for Processor Failures**

| <b>Event</b>                                                         | <b>Risk Mitigation Process/Description</b>                                                                                                                                                                                                                                                                                                                                                       | <b>P<sub>o</sub> w/o Risk Mitigation</b> | <b>P<sub>o</sub> with Risk Mitigation</b> | <b>Criticality</b> |
|----------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------|-------------------------------------------|--------------------|
| Single string FMS fails                                              | Use higher quality hardware; maintenance should follow recommended/certified practices and schedule; use redundant processors and multiplex architectures, if possible. Have different functional levels run on different processors. Set up “fall back” procedures so that FMs “rebooted” on shared processors with other functional levels. Require operation in designated combat zones only. | L                                        | L to VL                                   | A - B              |
| For multiplex: RM fails to correctly ID processor faults             | Use certified RM and standard FDI practices.                                                                                                                                                                                                                                                                                                                                                     | VL to EL                                 | EL                                        | A - B              |
| Summary from above: FMS solution not available - fails to deconflict | See above                                                                                                                                                                                                                                                                                                                                                                                        | VL to EL                                 | EL                                        | A-B                |

See Table E.5 for descriptions of the higher levels of the FTA diagram shown in Figure E.4.

#### **E.5.6 Fault Tree Analysis due to FMS Software Errors**

The FTA diagram shown in Figure E.7 isolates the fault pathway for algorithm, coding or convergence errors. Following the figure, Table E.8 gives a description of the risk mitigation processes required to bring the probability of the ownship colliding with another vehicle to an acceptable value for successful certification.



**Figure E.7. Fault Tree Analysis for Collision Avoidance Function due to Software Errors**

**Table E.8. Risk Mitigation Processes for Software Errors**

| Event                                                                                    | Risk Mitigation Process/Description                                                                                                                                                                                                                                                                                                                                                         | P <sub>o</sub> w/o Risk Mitigation | P <sub>o</sub> with Risk Mitigation | Criticality |
|------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------|-------------------------------------|-------------|
| OMM coding error                                                                         | Perform design time verification analysis to extent possible. Apply advanced formal methods at design time. Develop RTA checks for this occurrence. One approach may be to develop a run time variant of design time formal methods. Code should be constructed to allow for run time checks (akin to built-in self-tests, BISTs). Criticality can vary from A to E depending on the error. | L                                  | VL                                  | A - E       |
| OMM algorithm design error                                                               | Perform design time validation analysis to extent possible. Develop RTA checks for this occurrence. Nature of checks will depend on sub-function being checked. Code should be constructed to allow for run time checks (akin to built-in self-tests, BISTs). Criticality can vary from A to E depending on the error.                                                                      | L                                  | EL?                                 | A - E       |
| OMM non-convergence                                                                      | Develop processes to “continue through” non-convergence time (e.g. continue to deliver last good solution, or have “bank” of pre-calculated solutions. Develop RTA checks for this occurrence. Code should be constructed to allow for run time checks (akin to built-in self-tests, BISTs). Criticality depends on how long non-convergence continues.                                     | L                                  | VL                                  | A - B       |
| Summary from above: OMM solution in error due to software errors and fails to deconflict | See above                                                                                                                                                                                                                                                                                                                                                                                   | L                                  | VL to EL                            | A - E       |

See Table E.5 for descriptions of the higher levels of the FTA diagram shown in Figure E.4.

## E.6 Summary

In summary, RTA may be most useful in determining algorithm or coding errors and performing the last possible collision avoidance check for overall safety. RTA may have less value under hardware sensor or actuator failures. For hardware failures, risk mitigation practices are well matured and should be followed to the required level of safety for certification.

From the above analysis, it is evident that two main areas for potential failures to occur are:

1. Synchronization issues: since we are concerned with a fleet of vehicles in a cooperative control architecture, key to a successful mission will be accurate and timely information

exchange. However, communication latencies/dropouts and general sensor errors and measurement inaccuracies often occur, especially in a battlefield environment (e.g. dust, smoke, signal jamming, etc. can cause imperfect operating conditions). This can lead to errors being propagated through the solution process. With continuous *negotiating* between each ownship's MPS and FMS and, in turn, continuous *negotiating* between fleetmates' MPSs, there may be significant probability of timing issues occurring, whereby race conditions or deadlock conditions arise, leading to unforeseen, odd behavior in the overall management of the fleet. Or, an ownship's MPS/FMS pair may have *fallen behind* in their solution and are not operating on current information, while its fleetmates' MPS/FMSs may be incorrectly assuming that it is *keeping up*.

2. Non-Convergence issues: due to the complexity of the algorithms in both the MPS and FMS, there may always be the possibility of non-convergence, depending on the approaches under consideration. This too can lead to solutions *falling behind* and violating framerates.

A proper design approach for advanced, intelligent and learning algorithms in this context should be to add in robustness to such failure conditions. For example, the system should be able to recognize erroneous behavior and be able to easily and quickly *reset* the algorithms as often as needed. Or, there should be temporary backup procedures or plans of action when the system cannot converge to a solution due to unforeseen environmental conditions or unexpected scenarios.

Although the above description seems similar to run time assurance, it is not. It is the process of advanced systems performing *self-analysis* in advanced ways, with advanced recovery actions. These procedures should all take place *before* safety becomes an issue. Run time assurance is focused strictly on safety – when all else has failed, then the RTA system should take over.

## Appendix F

### Feasibility of Computing Switching Condition for No-Fly Zone Avoidance and Flight Safety Using Formal Methods Tools

#### F.1 Introduction

This chapter investigates the feasibility of computing the switching condition for no-fly zone avoidance and flight safety for a UAS using state-of-the-art formal methods tools, specifically, model checkers for hybrid systems. A *model checker* is a tool for constructing and analyzing the state-space of a given system. A model checker can be used, for example, to determine whether all of the reachable states of a given system satisfy a safety requirement. A *hybrid system* is a system with both discrete and continuous variables. A transition relation specifies the transitions between different values of the discrete variables. For each combination of values of the discrete variables, there is a collection of differential equations that describe the evolution of the continuous variables. The UAS model used in this investigation is based on the UAS guidance model presented earlier in this report.

For simplicity, this investigation considers only the case of a single no-fly zone directly ahead of the UAS. No-fly zones in other positions could be handled similarly. The absolute orientation of the vehicle is unimportant, so we assume it is initially moving in the positive X direction. With these assumptions, computation of the switching condition involves two kinds of computations, corresponding to the questions:

- 1) Is an unrecoverable state reachable from the current state for the UAS with the advanced controller, modeled as completely non-deterministic, in the RTA decision period (i.e., the time between switching decisions)?
- 2) Is an incorrect state reachable from the current state for the UAS with a safety controller that turns around as “quickly” (i.e., in as little X distance) as possible, in the time it takes the UAS to turn around (i.e., for its X velocity to become 0)? When the vehicle has turned around, we know that it has successfully avoided the no-fly zone.

Our experience and conclusions are summarized in Section F.2. In short, reachability computations for question 1) are possible with reasonable running time and accuracy. Reachability computations for question 2) are not feasible, due to the much longer time horizon, but question 2) can reasonably be answered using simulations. Therefore, computation of the switching condition is feasible using a combination of hybrid system reachability computations (performed by a hybrid system model checker) and simulation. Recommendations for future directions appear in Section F.3. Details of our experience with these two kinds of reachability computations appear in Sections F.4 and F.5, respectively.

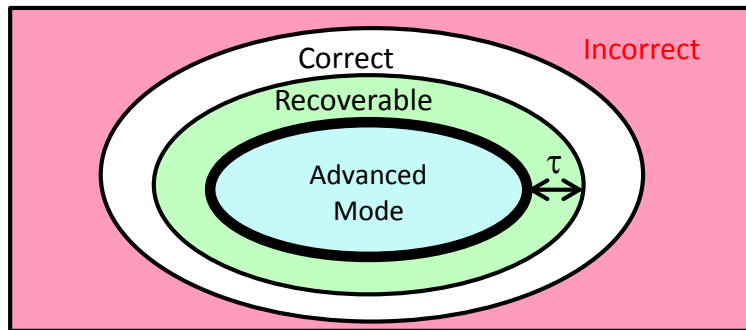
For expediency, since the goal is to evaluate feasibility, this investigation does not include a complete computation of the switching condition. The conclusions are based on experiments with selected computations representative of the full set of computations needed to compute the switching condition.



Switching conditions are often characterized in terms of a backward reachability computation from the set of incorrect states. However, that characterization requires an unbounded-duration backward reachability computation, which would be infeasible for this system. Using a forward-reachability characterization of the switching condition allows us to use finite-duration reachability computations, based on the above observation that, once the vehicle has turned around, we know that it has successfully avoided the no-fly zone, so the forward reachability computation can be terminated at that point.

There are several “flight safety” conditions that should be ensured by either the RTA system or other safety systems, e.g., that the velocity remains within reasonable limits. It is more efficient to consider these flight safety conditions together with no-fly zone avoidance, rather than separately, because this enables us to restrict attention to states satisfying all of these conditions, as discussed in more detail below.

A reasonable decision time for guidance-level properties for a relatively slow-flying vehicle like the UAS in the challenge problem is about 1 second, so the reachability computation is considered to be feasible if it can be done for at least 1 second of simulation time (we use the conventional term “simulation time” even though “reachability time” would be more accurate).



**Figure F.1. Illustration of Relationships between Advanced Mode and other States**

### F.1.1 Candidate Model Checkers

We evaluated the applicability of several model checkers for hybrid systems to these two kinds of reachability computations. A few of them are described briefly in this section. Some are not applicable to either problem because they do not support the non-linear dynamics of the UAS. Some are not applicable to advanced-controller reachability computations because they do not support inclusion flows, also called non-deterministic derivatives, i.e., differential inequalities that bound the derivatives, instead of differential equations that specify them exactly. We found and experimented with three tools that support the necessary non-linear dynamics and inclusion flows: Flow\*, HSolver, and HyCreate2. It is interesting to note that, although they solve essentially the same problem (with minor variations), they use very different algorithms. Also, they are based on conceptually similar models of hybrid systems, so it is relatively easy (for a human) to translate a model from one tool’s modeling language to the others, despite syntactic differences.

**Flow\*** (<http://systems.cs.colorado.edu/research/cyberphysical/taylormodels/>) is a hybrid system model checker developed primarily by Xin Chen. Flow\* supports inclusion flows of the form

$e+c_1 \leq \dot{x} \leq e+c_2$ . Flow\*’s reachability algorithm is described in [Chen 2012]. For each timestep, it computes an approximate solution to the differential equations during that time interval, starting from a given region of initial states. The approximate solution is represented as a Taylor model. This is similar to the well-known use of Taylor expansions to approximate a function near a given point. A Taylor model is a polynomial approximation that describes how a region in the state space evolves during a time interval. At each instant during that interval, it defines a region in the state space whose boundaries are polynomial surfaces; this is potentially more accurate than simply using boxes, i.e., regions in the state space whose boundaries are planes. Specifically, a Taylor model consists of: (1) local variables  $\vec{v} \in [-1,1]^d$  and  $t \in [0,\tau]$ , where  $\tau$  is the duration of the timestep, and  $d$  is the number of dimensions, i.e., the number of state variables, (2) a polynomial  $p_x(t, \vec{v})$  for each state variable  $x$ , and (3) a remainder  $R_x$  for each state variable  $x$ . The remainder bounds the error between the exact solution and the approximate solution embodied in the polynomials. For each combination of values of  $\vec{v}$ ,  $p_x(t, \vec{v})$  is a possible value of  $x$  at time  $t$ . The user specifies the desired degree of the polynomials, the desired time step  $\tau$ , etc. Flow\* computes the states reachable in a specified amount of time by starting with a Taylor Model containing the initial states, computing a Taylor model  $tm_1$  representing the behavior of the system during the first timestep, using the region defined by  $tm_1$  at time  $\tau$  as the initial region of the computation for the second timestep, and so on. If, at any step, the remainder exceeds a specified bound on the acceptable error, Flow\* terminates the computation.

**HSolver** (<http://hsolver.sourceforge.net/>) is a model checker for hybrid systems developed by Stefan Ratschan and collaborators. HSolver supports general inclusion flows of the form  $e_1 \leq \dot{x} \leq e_2$ . A distinguishing feature of HSolver is that it does not use a fixed grid size or a timestep. It adaptively divides the state space into boxes of varying size. HSolver’s reachability algorithm is based on constraint solving. It computes a coarse initial approximation of the reachable states using large boxes, and using a constraint solver to determine whether there is a transition between two boxes, i.e., whether the system can evolve from a state in the first box to a state in the second box without passing through other boxes. HSolver repeatedly increases the accuracy by splitting boxes in selected areas of state space, near the boundary between the reachable and unreachable states; this process is called *abstraction refinement*. The computation terminates when a specified level of accuracy (based on the sizes of the boxes) is reached.

**HyCreate2** is a hybrid system model checker developed by Stanley Bak (stanley.bak.1@us.af.mil). HyCreate2 supports general inclusion flows of the form  $e_1 \leq \dot{x} \leq e_2$ . The state space is divided into boxes (also called *hyperrectangles*) using a fixed-size grid. The user specifies the desired grid size for each dimension, the desired width of the neighborhood (this indirectly determines the time step), etc. HyCreate2’s reachability algorithm is based on a method called *face lifting*. HyCreate2 starts with boxes containing the initial states. At each time step, compute newly reachable boxes by propagating faces of the existing boxes forward in time, based on the timestep and extremal values of derivatives of the state variables in a neighborhood of the face. By default, HyCreate2 assumes that, for a given box, the extremal values of the derivative of each variable are attained at corners of the box. If the extremal values may occur at any other point in the box, the user must specify those points. This is required for our UAS guidance model, due to the trigonometric functions in the differential equations. If a box exceeds a user-specified size threshold (because the derivatives are large), and if the *large*

*rectangle splitting* option was selected, HyCreate2 splits the box into smaller boxes, based on the grid size.

### **Tools that do not support inclusion flows**

The following tools support non-linear dynamics but not inclusion flows.

**dReach** (<http://dreal.cs.cmu.edu/dreach.html>) is a model checker for hybrid systems developed by Sicun Gao and collaborators at Carnegie-Mellon University. dReach, unlike the other hybrid system model checkers we considered, does not compute a representation of the set of reachable states. dReach is designed to determine whether a state (or sequence of states) satisfying specified conditions is reachable within a specified time limit. To do this, dReach constructs a large formula that contains a subformula for each timestep, characterizing the states reachable at that timestep, conjoins that large formula with the conditions characterizing the states of interest, and then checks satisfiability of the resulting formula. Sicun Gao said he plans to support inclusion flows in a future version of dReach.

**Ariadne** (<http://trac.parades.rm.cnr.it/ariadne/>) is an extensible library for analysis of systems described by hybrid automata. The core reachability algorithm in Ariadne uses grids to represent sets of reachable states at specific points in time and uses Taylor models (also called *flowpipes*) to represent how those sets of states evolve over time [Benvenuti 2008]. The algorithm divides the state space into a rectangular grid, where the grid size is determined dynamically based on specified precision bounds. At each step, flowpipes starting from previously reached cells are computed, and cells that intersect with these flowpipes are marked as reached. The fact that Ariadne “snaps” the Taylor models onto a grid, and Flow\* doesn’t, is probably the largest difference between them. Ariadne does this to promote termination when computing the states reachable in an unbounded amount of time. In the experiments reported in [Chen 2012], Ariadne is unable to handle several models that Flow\* handles successfully, due to an excessive number of grid cells, but Ariadne is faster than Flow\* for models it is able to handle. Ariadne’s developers did not reply to inquiries about extending Ariadne to support inclusion flows.

**C2E2** (<https://publish.illinois.edu/c2e2-tool/>) incorporates a novel simulation-based reachability algorithm. C2E2 requires users to specify a *discrepancy function*, which might be difficult in practice. C2E2’s developers are working on techniques to compute discrepancy functions automatically. C2E2’s developers said they plan to support inclusion flows in a future version of C2E2.

## **F.2 Summary of Experience and Conclusions**

In theory, application of the model checkers to this problem should be automatic. In practice, there are two potential problems: scalability and usability. In summary, we conclude that computation of the switching condition is feasible for our case study, using a combination of state reachability computations and simulations.

For our case study, the advanced-controller reachability computations are feasible with HyCreate2, and would take about 3.5 hours of CPU time on a single core. If multiple cores or multiple CPUs are available, the computation can easily be parallelized, to reduce the end-to-end computation time to a few minutes. Furthermore, comparison with results from simulations show that the results are tight, i.e., contain relatively few unreachable states. These results are

based on a decision period (and hence time horizon) of 1 second. We did not do experiments with longer time horizons but tentatively expect that the computations would be feasible for decision periods of a few seconds, perhaps up to several seconds.

However, the safety-controller reachability computations are not feasible with any of these tools, due to lack of scalability. Specifically, the relatively large number of state variables (namely, 9), the relatively long time horizon (40+ seconds), and the complexity of the differential equations together make the computation beyond the capabilities of the current tools on current computing hardware. Depending on the settings, either the computation proceeds at a reasonable speed but too much inaccuracy accumulates as it progresses, or the computation proceeds at an unacceptably slow speed.

On the positive side, the overall computation of the switching condition is feasible, because simulation, instead of state reachability computations, can be used to explore the behavior of the UAS with the safety controller. Specifically, simulations from selected states in each initial region can be used instead, to determine whether the region contains any unrecoverable states. The selected states need to include any state that might be a “worst case” with respect to the correctness property (in our case study, the correctness property is keepout zone avoidance and flight safety). The selected states usually include the corners of the region and possibly some interior states; for example, if the corners of the region have positive and negative values for an angle such as FPA, interior states with FPA=0 might need to be included. Determining which interior states need to be considered can be difficult in general but should be feasible for systems such as the UAS with safety controller. The task is simplified by the fact that the safety controller is deterministic. In contrast, it would be much more difficult to use simulations, instead of reachability computations, to analyze the behavior of highly non-deterministic systems, such as the UAS with the non-deterministic model of the advanced controller, because it requires identifying the worst-case *behaviors* of the system’s non-deterministic components with respect to the property being evaluated, and implementing them in deterministic versions of those components that can be used in simulations.

## F.3 Future Directions

### F.3.1 Improvements to State Reachability Tools

**Scalability.** A state-of-the-art tool, HyCreate2, was able to handle the challenging reachability analysis for the UAS with the non-deterministic model of the advanced controller in our case study, but improved scalability will clearly be needed. If the time horizon were increased by several seconds, or several more state variables were added to the dynamic model of the UAV, it is unclear whether the current tool would be able to handle it. Generally, the scalability of state reachability tools for hybrid systems can be improved by finding new ways to incorporate the key ideas that have greatly improved the scalability of state reachability tools for discrete systems, namely, abstraction, symbolic representations, symmetry reduction, compositional reasoning, and partial-order reduction.

An example of abstraction (mentioned by Stanley Bak) is, instead of analyzing a non-linear system directly, to construct and analyze an *upper linearization* of it (i.e., a hybrid system with linear dynamics in each mode, and whose reachable states include all reachable states of the given non-linear system) and, depending on the properties of interest, possibly also a *lower*

*linearization* of it (i.e., a hybrid system with linear dynamics in each mode, and whose reachable states are a subset of the reachable states of the given non-linear system). This technique is useful, because linear systems can be analyzed more efficiently than non-linear systems. This technique is effective when the resulting over-approximation (or under-approximation) is not too severe. The accuracy—and analysis cost—can be increased by increasing the number of modes; the linear dynamics in each mode models a different region of the non-linear dynamics. Another type of abstraction is to replace a complex state representation with a simpler but more approximate one. For example, in HyCreate2, this might be done by replacing a set of overlapping boxes with a single box that encloses them (this is a simple form of convex hull).

Symbolic representations (e.g., boxes, Taylor models) are already used in all state reachability tools for hybrid systems, to represent regions in the continuous dimensions of the state space, but there may be room for improvement by using more sophisticated symbolic representations. For example, modifying HyCreate2’s reachability algorithm to use  $n$ -dimensional polytopes instead of  $n$ -dimensional boxes would potentially yield more accurate results for many systems, at the cost of increasing the complexity of the algorithm and the running time of each analysis step.<sup>3</sup>

Decomposing the initial region and separately (or mostly separately) exploring the reachable states from each initial sub-region can improve accuracy and scalability. Some tools, such as HyCreate2, already do this automatically; others, such as Flow\*, do not.

A complementary approach to improving scalability is to design systems for ease of analysis, when possible. For example, systems that are more deterministic have fewer reachable states and hence are easier to analyze. Similarly, systems that are strongly stable are easier to analyze, both because they tend to have fewer reachable states, and because the system’s convergence counteracts the reachability computation’s over-approximation: the former shrinks the fringe of states currently being explored, while the latter expands it.

There are also a few specific ideas to improve scalability of HyCreate2 that are not implemented yet. One is *selective splitting*, i.e., splitting large rectangles on selected dimensions, instead of on all dimensions simultaneously. Another is to more aggressively limit the computation to states satisfying a user-specified invariant; states not satisfying the invariant are considered infeasible or uninteresting and should be dropped, even if they are reachable according to the dynamics. Currently, HyCreate2 discards boxes containing no states satisfying the invariant, but keeps an entire box if the box contains any states satisfying the invariant. This can be improved by discarding parts of the box not satisfying the invariant. This optimization can improve the accuracy and the running time. Flow\* already incorporates this optimization: it computes the intersection of each computed Taylor model with an invariant expressed as a conjunction of polynomial inequalities.

---

<sup>3</sup> Some hybrid systems reachability algorithms that use polytopes are described in: Oded Maler, Algorithmic Verification of Continuous and Hybrid Systems. In *Proceedings of the 5th International Workshop on Verification of Infinite-State Systems (INFINITY 2013)*, volume 140 in *EPTCS*, 2013. <http://arxiv.org/abs/1403.0952v1>

Comparing the scalability of different tools is difficult, because scalability depends heavily on each tool’s settings, and it is difficult for non-expert users to know whether they found the best—or even pretty good—settings. This segues into the next topic...

**Auto-Tuning.** Automatic adjustment (tuning) of the settings that affect the speed and accuracy of the reachability computation would be extremely valuable. Tuning took considerable effort for us as new users of the tools, even though we understood the underlying reachability algorithms based on descriptions in the tool documentation and research papers. Tuning would take non-negligible effort even for experienced users. Ideally, the user would specify a high-level goal, such as “get the most accurate result possible in at most 1 hour on an 8-core computer”, and auto-tuning would adjust all of the settings to meet that goal. Note that auto-tuning may be done statically or, in some cases, dynamically. For example, Flow\* already supports a limited form of dynamic auto-tuning: the user can specify a range of values for some settings (e.g., timestep), and Flow\* dynamically adjusts the value within that range based on the current remainder estimate. The version of HyCreate2 used in our experiments did not currently incorporate any static or dynamic auto-tuning, but the newest version has some support now for automatic detection of the grid size and face lift ratio. One can imagine extending HyCreate2 to dynamically adjust settings such as the grid size; this is roughly analogous to adaptive mesh generation in finite element methods.

**Manual Tuning.** Until auto-tuning is available, manual tuning would be facilitated by more detailed guidelines and by more detailed feedback about the computation. For example, in Flow\*, the user must specify a separate remainder estimate for each state variable, but Flow\* reports only whether a bound on the remainder is exceeded. Flow\* does not provide any per-variable feedback about the remainder. Consequently, it is difficult to decide which variables’ remainder estimates to adjust when the bound is exceeded. As another example, in HyCreate2, the user specifies a regrid ratio for each variable, but the user does not receive feedback on the number of regriddings triggered by each variable. Also in HyCreate2, information about values of the derivative of each variable (e.g., average over all explored boxes of the maximal value of the derivative of the variable when lifting the corresponding face of the box, or, even better, a histogram instead of an average) might be useful when adjusting the face size ratio for each variable.

**Simulation.** Simulation is valuable to explore and understand the behavior of a model. We found HyCreate2’s simulation capability very helpful. HSolver and Flow\* do not support simulation. Simulation needs to deal with non-determinism, including discrete non-determinism, which arises when transitions to multiple modes are possible, and continuous non-determinism, which arises from inclusion flows and when a mode transition can occur at any time within some time interval. HyCreate2 allows the user to choose policies that specify which of the possible behaviors will be explored; a simulation is performed for each of those behaviors, and the results are aggregated in a single visualization.

**Modeling Language.** A common limitation of the modeling languages of Flow\*, HyCreate2, and HSolver, inherited from the hybrid automaton framework on which they are based, is that the discrete part of the state is limited to a single variable, called the mode, which ranges over a predefined finite set of atomic values. This limitation did not affect our case study but would be inconvenient for more complex models, especially if they involved dynamically allocated data

structures such as lists, stacks, or trees. Stanley Bak suggests that one approach to generalizing the framework is, in the style of synchronous languages such as Lustre, to introduce a richer language to express the sequential computations, and to regard an execution of the system as (nearly) instantaneous executions of sequential computations expressed in this language interleaved with continuous evolution of the state according to given differential equations.

Another common limitation of the modeling languages (later eliminated from HyCreate2, at our suggestion) is the requirement to use literal constants in certain places in the model, for example, as the initial value of each variable. We circumvented this limitation in the other tools by using a preprocessor, gpp, to invoke a command-line calculator program, bc, to do constant folding, i.e., to replace expressions with their values. However, preprocessing is slightly cumbersome and complicates debugging of syntax errors.

Another limitation of the modeling languages, except HyCreate2's, is support for a limited set of mathematical functions. For example, Flow\* does not support arctangent, which is used in another case study that we started to consider, namely, wing morphing. HyCreate2 is unique among these tools in that the mathematical functions are written directly in Java, rather than a limited language defined by the tool, so arbitrary mathematical functions are available. This approach is feasible in HyCreate2 because it performs only numerical evaluation of these functions; it never performs symbolic manipulation of them. This approach cannot easily be adopted by Flow\* or HSolver, because they perform symbolic manipulation (and numerical evaluation) of functions. Nevertheless, their modeling languages can be extended to support additional mathematical functions useful in aerospace models, such as inverse trigonometric functions.

**Visualization.** HyCreate2 and HSolver can show a visualization of the reachability computation while it is in progress. HyCreate2, Flow\*, and HSolver can generate a visualization of the result of a completed reachability computation. The in-progress visualizations are limited to a single 2D projection of the explored states; support for visualization of multiple 2D projections would be helpful, to allow more detailed monitoring of in-progress computations, especially long-running ones. The visualizations of completed computations are also limited to 2D projections. When dealing with models of aerial vehicles, 3D plots are sometimes desired. We wrote scripts that read textual output files from HSolver, Flow\*, and HyCreate1 and use GNU Octave, an open-source MATLAB-like application, to produce 3D plots. It would be convenient if the tools directly supported 3D visualizations.

### F.3.2 Further Evaluation of State Reachability Tools

**Additional Case Studies.** Our findings, especially about scalability of the tools to handle reachability problems that arise in RTA design for UASs, are currently based on a single case study. Additional case studies should be done to confirm or revise these preliminary findings. For example, it would be interesting to know whether HyCreate2 can handle the model of a UAS with wing morphing, controlled by the phase 1 transition controller (this model was developed in another part of this project). The desired time horizon for that analysis is only a few seconds, and the model is deterministic or, if non-deterministic wind is considered, slightly non-deterministic. In this regard, the wing morphing model will be significantly easier to analyze than the UAS with a highly non-deterministic model of the advanced controller; our experiments with the existing case study confirmed the expectation that reducing the level of non-



determinism made the reachability computations easier (e.g., longer time horizons could be handled). On the other hand, the equations of motion in the wing morphing model are more complex than the equations of motion in the existing case study. Case studies for complex systems that employ run-time assurance based on the Simplex architecture for multiple components would also be interesting.

**Statistical Model Checking.** A *statistical model checker* (SMC) provides probabilistic guarantees (in contrast to unconditional guarantees) about the behavior of stochastic models of hybrid systems, by statistically sampling and evaluating the possible behaviors of the system. The SMCs for hybrid systems we know of do not support inclusion flows, so they cannot be used to analyze UASs with completely non-deterministic models of advanced controllers, but they could be used to analyze deterministic or finitely non-deterministic models of UASs with safety controllers, if the model has a stochastic aspect, e.g., if a probability distribution over the initial states is given. A candidate tool is the SMC extension to UPPAAL (<http://people.cs.aau.dk/~adavid/smc/>); it supports non-linear differential equations with polynomials and trigonometric functions.

#### F.4 States Reachable With Advanced Controller

The goal of reachability computations with the advanced controller is to determine a set of states *AdvancedMode* such that, starting from any of those states, the system is guaranteed to remain in recoverable states for the RTA decision period. It is safe for the RTA system to allow the advanced controller to be in control of the system in those states. Table F.1 illustrates the relationship between *AdvancedMode* and some other sets of states;  $\tau$  is the RTA decision period, and correct states are states in which the UAS is not in a no-fly zone and the flight safety invariants are satisfied.

The process of computing the switching condition starts with an initial guess for *AdvancedMode*. That set is then partitioned into regions (e.g., boxes). For each of those regions  $R$ , a reachability computation is performed to determine whether some unrecoverable state is reachable within the RTA decision period. If not,  $R$  is added to *AdvancedMode*; otherwise,  $R$  is excluded from *AdvancedMode* (or  $R$  may be split into smaller regions, and each of the resulting regions may be tested). Although we could instead test the entire initial guess at once, the resulting reachability computation might be too expensive: the cost increases with the size of the initial region (as well as the time limit). Also, testing the entire initial guess would provide only a yes/no answer; testing regions of the guess separately returns a subset of the initial guess that satisfies the requirements for *AdvancedMode*.

Note: A similar approach based on simulation can be considered. A grid of states spaced across the initial guess for *AdvancedMode* is chosen. A simulation from each state in the grid is used to determine whether an unrecoverable state is reachable from that state within the RTA decision period. If not, some region around that state is added to *AdvancedMode*; otherwise, some region around the state is excluded from *AdvancedMode*. However, in general it is not easy to choose these regions in a rigorous way to obtain formal guarantees, especially when the system is highly non-deterministic, and its behavior is consequently harder to understand and analyze.



**Table F.1. Variables, Invariants, Initial State, Initial Guess for AdvancedMode**

| <b>Var-<br/>iable</b> | <b>Meaning</b>                                                     | <b>Limits in Flight<br/>Safety Invariant</b>     | <b>Value in<br/>State S0<br/>(approx.)</b> | <b>Minimum of Range<br/>in Initial Guess for<br/>AdvancedMode</b> | <b>Maximum of<br/>Range in Initial<br/>Guess for<br/>AdvancedMode</b> |
|-----------------------|--------------------------------------------------------------------|--------------------------------------------------|--------------------------------------------|-------------------------------------------------------------------|-----------------------------------------------------------------------|
| T                     | Thrust (pounds)                                                    | Tmin=10<br>Tmax=200                              | T0=30.9                                    | 0.7*T0 = 22                                                       | 1.6*T0 = 50                                                           |
| L                     | Lift (pounds)                                                      | Lmin=0<br>Lmax=KLmax*V<br>max <sup>2</sup> =2013 | L0=1042.9                                  | 0.7*L0 = 730                                                      | 1.6*L0 = 1668                                                         |
| BA                    | Bank Angle (rad)                                                   | BAmin=-0.35<br>BAmax=0.35                        | BA0=0                                      | 0                                                                 | BAmax = 0.35                                                          |
| V                     | Velocity (feet/sec)                                                | Vmin=88<br>Vmax=176                              | V0=127.6                                   | 0.7*V0 = 89                                                       | 1.3*V0 = 165                                                          |
| FPA                   | Flight Path Angle<br>(rad)                                         | FP Amin=-0.70<br>FP Amax=0.52                    | 0                                          | FP Amin = -0.7                                                    | FP Amax = 0.5                                                         |
| HA                    | Heading Angle (rad)                                                |                                                  | 0                                          | 0                                                                 | 0                                                                     |
| X                     | X position (feet)                                                  |                                                  | 0                                          | 0                                                                 | 0                                                                     |
| Y                     | Y position (feet)                                                  |                                                  | 0                                          | 0                                                                 | 0                                                                     |
| H                     | H position (feet)                                                  |                                                  | 1000                                       | 1000                                                              | 1000                                                                  |
| xT                    | Variable in thrust<br>control law of safety<br>controller (pounds) |                                                  | Same as T                                  | Same as T                                                         | Same as T                                                             |
| xL                    | Variable in lift<br>control law of safety<br>controller (pounds)   |                                                  | Same as L                                  | Same as L                                                         | Same as L                                                             |

For each region, it is not necessary to determine the exact set of unrecoverable states reachable within the RTA decision period. Therefore, we can limit the reachability computation to the recoverable states, and simply report “unrecoverable states are reachable” if the boundary of the set of recoverable states is encountered. This can significantly reduce the cost of the reachability computation, if many unrecoverable states are reachable. If an exact characterization of the set of recoverable states is complex (so it would be inconvenient to limit the reachability computation to exactly that set), we can instead limit the reachability computation to some convenient superset of the recoverable states, check whether the reported reachable states contain any unrecoverable states, and if so, report that unrecoverable states are reachable.

The set of recoverable states is determined by reachability computations with the safety controller (i.e., questions of type (2) introduced in Section F.1). Since we have not yet performed all of those calculations, we instead limit the reachability computations with the advanced controller to a convenient superset of the recoverable states, namely, the set of correct states (see above diagram), i.e., states that satisfy all of the properties that the RTA system or other safety systems are supposed to ensure.

The more correctness properties we consider, the smaller the sets of correct states and recoverable states. This is the benefit of considering flight safety properties together with no-fly zone avoidance. We consider the following flight safety properties:  $T \geq T_{min}$ ,  $T \leq T_{max}$ ,  $L \geq L_{min}$ ,  $L \leq KL_{max} * V^2$ ,  $BA > BA_{min}$ ,  $BA \leq BA_{max}$ ,  $V \geq V_{min}$ ,  $V \leq V_{max}$ ,  $FPA \geq FPA_{min}$ ,

and  $FPA \leq FPA_{max}$ . We did not include an invariant corresponding to entering the no-fly zone, but this would be legitimate.

We use the following initial guess for AdvancedMode. It was obtained by starting from a state  $S0 = \langle T0, L0, BA0, V0, \dots \rangle$  similar to the initial state in the MATLAB simulation of the UAS guidance model, and expanding the possible values of each variable to a range.

Notes: (1) By symmetry, we do not need to consider  $BA < 0$  in the initial states; it is symmetric to  $BA > 0$ , provided the safety controller uses  $BACmd=B_{Amin}$  instead of  $BACmd=B_{Amax}$  when it takes over in a state with  $BA < 0$ . (2) For  $V$ , an upper limit of  $1.3 * V0$  (instead of  $1.6 * V0$  as for other variables) is used, because  $1.4 * V0 > V_{max}$ . (3)  $xT$  and  $xL$  are variables of the safety controller and hence are not used in experiments with the advanced controller described in this section; they are used in experiments with the safety controller, described in the next section. (4) The absolute position and orientation of the UAS are not important, so without loss of generality, we assume the UAS is initially at position  $\langle 0, 0, 1000 \rangle$  and moving in the positive  $X$  direction. (5) The invariant  $L_{max} \leq KL_{max} * V^2$  imposes a velocity-dependent bound on lift.

Consequently, some combinations of  $L$  and  $V$  in this initial guess for AdvancedMode are infeasible, e.g., at the corner with the highest value of  $L$  and the lowest value of  $V$ . Since the invariant is enforced in the reachability calculation, including some infeasible states in the initial region has a limited effect on the reachability calculation, and does not affect the conclusions of the report. If we wanted to actually compute the switching condition, we would refine the initial guess for AdvancedMode to exclude infeasible states, by making it a suitable collection of smaller boxes, instead of a single large box.

To partition this initial guess for AdvancedMode into regions, we divide the above range for each variable into  $numRange$  sub-ranges. We initially took  $numRange = 3$ . This leads to  $3^5 = 243$  regions (it is  $3^5$ , not  $3^7$ , because  $xT$  and  $xL$  are not used by the non-deterministic controller). It is probably possible to reduce the number of regions by exploiting correlations between values of variables. For example, if states in which  $xT$  and  $T$ , or  $xL$  and  $L$ , have widely different values are unrealistic, then we could exclude regions in which those variables are not in the same (or perhaps consecutive) ranges.

Although this is a large number of regions, reachability computations for different regions can be done in parallel. On, say, a small cluster with 20 quad-core processors, the end-to-end running time would be about 3 times the running time for a single reachability computation. This amount of computing power could be rented from a cloud provider when needed.

The use of only 3 sub-ranges for each variable is rather coarse, so unnecessarily large regions of states may be excluded from AdvancedMode. To address this without increasing the total number of regions too much, after the reachability computation has been done for each region, regions on the “boundary” of AdvancedMode (i.e., regions that are not in AdvancedMode but are adjacent to regions in AdvancedMode) can be split into smaller regions, and a reachability computation can be done starting from each of those regions.

To evaluate feasibility, we select a few representative regions and perform reachability computations for them. We mainly used a region  $R_{ctr}$  near the center of the ranges in Table F.1, specifically, the region whose bottom corner is the initial state  $S0$ , and whose size (i.e., edge

lengths) is determined by numRange, defined in Section B.4.0.<sup>4</sup> The goal of the experiments is to evaluate the running time and accuracy of the reachability computations with each hybrid-system model checker. “Accuracy” here refers to whether the resulting set of states is a tight or loose over-approximation of the states actually reachable within the given time bound, as determined by simulation.

#### **F.4.1 Summary of Conclusions for Reachability with Advanced Controller**

Flow\* can achieve reasonable accuracy for up to 1 second of simulation time, but only if the region of initial states is not too large. It appears that reachability computations with numRange=3 are not feasible with Flow\*. With numRange=7, reachability computations for individual regions appear to be feasible if the RTA decision period is 1 second; a sample computation for one region takes about 18 min. However, with numRange=7, the total number of regions is  $7^5 = 16807$ , which is probably impractical. On, say, a small cluster with 20 quad-core processors, the end-to-end running time would be at least 63 hours; it is likely to be higher, because smaller (hence more) regions will probably be needed near the edges of the initial guess for AdvancedMode. Comparison of the results of the reachability computation with results from simulations for the maximum reachable velocity and X position show good accuracy (i.e., relatively tight upper bound) for velocity (upper bound from reachability computation is 151, compared to actual value of 148 from simulation) and less accuracy (i.e., looser upper bound) for X position (upper bound from reachability computation is 159, compared to actual value of 140 from simulation).

Flow\* seems to provide limited control over the trade-off between running time and accuracy. We were unable to find settings for Flow\* that allowed faster but less accurate computation than this. Recall that Flow\* maintains a “remainder estimate”, which is a bound on the difference between the exact solution to the differential equations and the approximate solution embodied in the Taylor models. If the remainder estimate gets too large, Flow\* aborts the computation, with the confusingly worded message “Remainder estimate not large enough” (RENLE).

HyCreate2 can successfully perform reachability computations for 1 second of simulation time with numRange=3. Furthermore, the calculation can be slow or fast, depending on the settings. The best results are obtained with a small grid and large rectangle splitting off: the computation takes 50 seconds, the bound on the maximum velocity is tight (163.6 compared to value of 161.9 from simulation), and the bound on the maximum X position is tight (155.7 compared to value of 154.8 from simulation). The end-to-end computation time for all 243 regions would be about 3.5 hours on a computer with a single core and only a few minutes on a cluster with 20 quad-core processors. With large rectangle splitting turned on, the running time for a sample region takes about 130 min for a slow calculation and a few seconds for a fast calculation. It is possible to achieve intermediate balances between speed and accuracy, since HyCreate2 allows fine-grained control over the calculation, although considerable experimentation with different settings might be needed. The bound on the maximum reachable velocity is fairly tight for the slower

---

<sup>4</sup> Depending on numRange, Rctr might not be exactly aligned with any of the regions obtained by partitioning the range for each variable into numRange sub-ranges and taking a cross-product of those sub-ranges to obtain regions. However, Rctr has the same size as those regions and overlaps with a few centrally located such regions.

computation (upper bound is 164, compared to actual value of 161.9 from simulation); the bound from the fast computation is looser (172). However, the bound on the maximum reachable X position is fairly loose for both computations (upper bound is 194 from the slow reachability computation, and 193 from the fast reachability computation, compared to actual value of 154.8 from simulation). HyCreate2's slow computation for one region whose size is based on numRange=3 takes considerably longer than Flow\*'s computation for one region with size based on numRange=7, but this is more than outweighed by the fact that numRange=3 produces so many fewer regions. The end-to-end running time on a small cluster with 20 quad-core processors would likely be on the order of 8 hours.

HSolver could be applied to this problem, but our earlier experiments with HSolver showed that HSolver is much slower than HyCreate2 for hybrid systems like the UAS, presumably due to the cost of constraint solving, so we did not experiment with it again in this investigation.

dReach does not support non-deterministic derivatives, so it is not suitable for analyzing the non-deterministic model of the advanced controller.<sup>5</sup>

#### **F.4.2 Exploring the Initial Guess for AdvancedMode**

To gain some insight into the behavior of the system at the extrema of the initial guess for AdvancedMode, Flow\* was used to calculate states reachable from the two extremal states: the state Smin having the smallest value of each variable, and the state Smax having the largest value for each variable. For these reachability computations only, a trivial controller was used that keeps T, L, and BA fixed at their initial values. For Smin, Flow\* stops immediately (after 0 or 1 time step), because all reachable states violate some conjuncts in the invariant hence get discarded. Specifically, the violated conjuncts are  $FPA \geq FPA_{min}$  and  $L \leq KL_{max} * V^2$ . The former is not surprising, because FPA starts at  $FPA_{min}$ , and FPA immediately decreases, because of the low thrust and velocity. To avoid this immediate termination, we did a reachability computation starting from the state obtained from Smin by giving L and V their maximum values in the initial guess in Table F.1. The reachability computation from this state proceeds for 0.54 sec of simulation time before it stops due to all states reachable at that time violating some conjuncts in the invariant. Similarly, for Smax, Flow\* stops immediately, because none of the reachable states satisfy the conjunct  $FPA \leq FPA_{max}$  is violated. If we modify Smax so that FPA is initially  $FPA_{min}$ , the reachability computation proceeds for 0.51 sec of simulation time before it stops due to all states reachable at that time violating some conjuncts in the invariant. These results provide some evidence that the UAS can fly correctly for at least a short time from states in the initial guess for AdvancedMode.

#### **F.4.3 Simulations Used to Evaluate Accuracy of Reachability Results**

For some variables, it is easy to construct a deterministic controller that maximizes the rate of increase of that variable. Simulations of the UAS with that controller can be used to determine

---

<sup>5</sup> Sicun Gao mentioned that dReach does allow variables to be initialized non-deterministically, and this can be used to model a variable with a non-deterministic but fixed derivative (e.g., introduce a variable Td initialized non-deterministically to a value between TdMin and TdMax, and take  $T' = Td$  and  $Td' = 0$ ). This is not equivalent to allowing the derivative to vary non-deterministically, but it might be a reasonable approximation in some cases.

the maximum value of that variable reachable from a specified initial region within a specified time. This value can be used to evaluate the accuracy of reachability results, i.e., to determine the looseness of the over-approximations. We did this for two selected variables: (1) the X position, because it is related to entering the no-fly zone, and (2) the velocity, because it is representative of variables constrained by a flight safety invariant. The simulations were performed using HyCreate2.<sup>6</sup>

To maximize V, we used a controller, called the *dive controller*, that sets  $T' = T_{dMax}$ ,  $L' = -L_{dMax}$ , and  $BA' = 0$ ; in other words, it dives with maximum thrust. HyCreate2 was used to simulate the UAS with this controller starting from all corners of a specified box. We also tried a variant of this controller that sets  $BA' = BA_{dMax}$ , but the maximum velocity was slightly lower.

To maximize X, we used a controller, called the *max-thrust level-flight controller*, that sets  $T' = T_{dMax}$ , uses the same control law as the safety controller for L (the safety controller is described in Section F.5.2), and sets  $BA' = 0$ . The control law for L attempts to reach and maintain a commanded velocity  $V_{cmd}$  and a commanded height  $H_{cmd}$ ; we take  $V_{cmd} = V_{max}$  and  $H_{cmd} = H_0$ . The justification for setting  $BA' = 0$  is that the maximum value of X will be achieved from a corner of  $R_{ctr}$  with  $HA = 0$  and  $BA = 0$  and with  $BA' = 0$ . (Recall that  $R_{ctr}$  is defined in Section F.4.0. If  $R_{ctr}$  did not contain a state with  $HA = 0$  and  $BA = 0$ , then we would need to use a control law for BA that attempts to reach and then maintain  $BA = 0$ .) Figure F.2 shows a graph of X, H for the simulations with this controller from the corners of region  $R_{ctr}$  with  $numRange = 3$ . We can see that it maintains level flight from some corners of  $R_{ctr}$ . We also tried the dive controller, but the maximum value of X it reaches is lower.

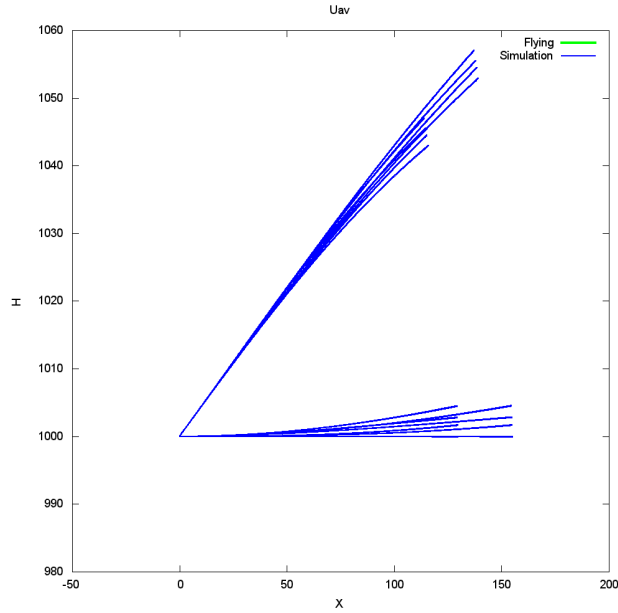
#### F.4.4 Feasibility Using Flow\*

All experiments involving Flow\* were performed by running Flow\* 1.2.0 on uav.model using the run script. Both of those files are available on VDL. Some of the initial experiments with Flow\* were run on a laptop with an Intel Core 2 Duo @ 2.4GHz. The remaining experiments were run on a desktop with an Intel Core 2 Quad @ 3.0GHz. Flow\* is single-threaded, so the number of cores does not affect the running time for a single reachability computation.

**Adjusting Flow\* Settings.** Table F.2 contains results of Flow\* reachability computations with various settings. An empty cell indicates the same value as the cell above it. Recall that the remainder estimation ratio, RER, is used in UAS.model to set the remainder estimation for each variable; specifically, the remainder estimation for a variable is the RER times the magnitude constant for the variable. “RENLE” abbreviates “remainder estimation not large enough”. All Flow\* computations use a timestep of .01.

---

<sup>6</sup> Implementation detail: An easy way to obtain the maximum value of a variable from the output file generated by a HyCreate2 simulation, is to set the HyCreate2 plot options to plot that variable on the Y axis, open the output file result/simulation.txt in Excel specifying space as the delimiter, and use a formula to calculate the maximum value in the second column.



**Figure F.2. Graph of X, H from HyCreate2 Simulations**

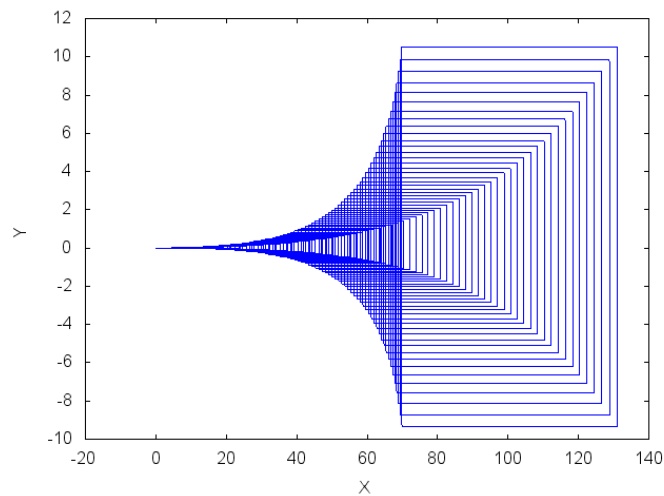
**Table F.2. Results of Reachability Computations for Various Settings in Flow\***

| Experiment # | RER  | Orders | Cutoff | numRange | Outcome          | Run Time (min:sec) |
|--------------|------|--------|--------|----------|------------------|--------------------|
| 1            | .001 | 3..6   | 1e-11  | 3        | RENLE @ 0.72sec  | 49:45              |
| 2            |      |        | 1e-9   |          | RENLE @ 0.72sec  | 38:11              |
| 3            |      |        | 1e-12  |          | RENLE @ 0.72sec  | 53:02              |
| 4            | .01  |        | 1e-7   |          | RENLE @ 0.75sec  | 11:34              |
| 5            |      |        | 1e-9   |          | RENLE @ 0.75sec  | 33:59              |
| 6            |      | 4..6   |        |          | RENLE @ 0.74sec  | 180:05             |
| 7            |      | 3..6   |        | 4        | RENLE @ 0.87 sec | 19:49              |
| 8            |      |        |        | 5        | RENLE @ 0.94 sec | 19:30              |
| 9            |      |        |        | 6        | RENLE @ 0.99 sec | 19:01              |
| 10           |      |        |        | 7        | RENLE @ 1.02 sec | 18:12              |

For all of the experiments summarized in the above table, the graphs of X, Y looked similar. A representative one appears in Figure F.3. The increasing spread in values over time mainly reflects the non-determinism of the advanced controller, not inaccuracy in Flow\*; as evidence of this, note that (1) we know from previous experiments with the UAS model that Flow\* can accurately analyze this model for tens of seconds of simulation time, and (2) we get similar results with a variety of different values for Flow\* settings that affect the accuracy of the computation.

We see from Table F.2 that the explored variations in these settings does not affect the simulation time until RENLE significantly. This is consistent with the results of earlier experiments with the same UAS model and a simpler initial region in which only the initial position varied. We did not experiment here with different timesteps, because decreasing the timestep in those experiments provided relatively small increases in the simulation time until

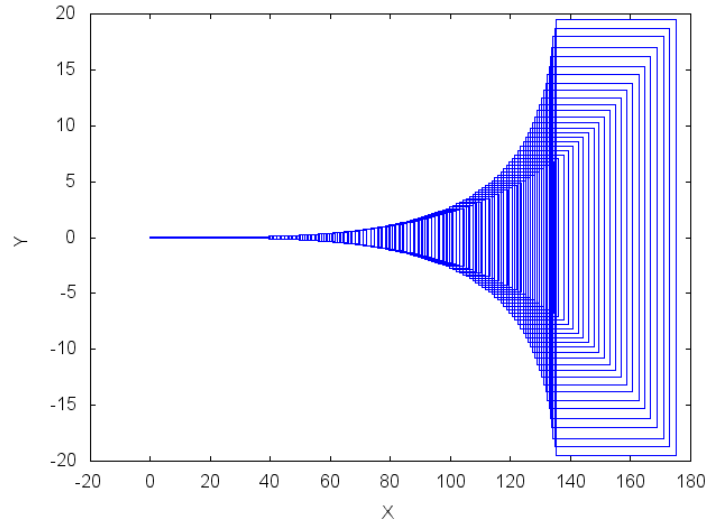
RENLE in return for relatively large increases in running time. We did not ask Xin Chen, Flow\*'s main developer, directly for help adjusting Flow\*'s settings for this model (partly because this model is not approved for public distribution), but we did ask him for such advice for our Flow\* model of wing morphing, for which Flow\*'s reachability computations also terminated early with RENLE. His general advice, for hybrid systems with 8 or more dimensions, is to use orders 3..4 (we use 3..6, but Flow\* usually keeps the order in the range 3..4 anyway, until just before RENLE), timestep  $\leq 0.01$  (we use 0.01) and cutoff  $1e-7$  (we use this and smaller values; a result with cutoff  $1e-9$  appears Figure F.3). He also commented that “most ODEs in practical models are stiff or nearly stiff. Therefore the overapproximations computed by Flow\* explodes in a short time.” In other words, for such systems, RENLE cannot be delayed significantly by adjusting Flow\* settings, except by using such large orders that the running time would be impractical.



**Figure F.3. Graph of X, Y from Reachability Computation from Region Rctr**

**Adjusting Size of Initial Region.** In previous experiments with Flow\*, the most effective way to delay RENLE was to use a smaller initial region. Therefore, we employed that approach here, too. First, to determine whether Flow\* can reach 1 sec of simulation time with a very small initial region, we did a reachability computation starting from a single state in Rctr, namely, S0. The reachability computation terminates with RENLE after 1.22 sec of simulation time, with running time 1:23. The graph of X, Y appears in Figure F.4. Next, we gradually increased numRange until at least 1 sec of simulation time could be analyzed. This occurred at numRange=7, as indicated in Table F.2.

**Accuracy.** The range for V generated by Flow\*, using the settings in the last row of Table F.2 is approximately 109 to 151. Based on HyCreate2 simulations for 1 sec from the corners of Rctr with numRange=7 using the dive controller described in Section B.4.3, the maximum reachable velocity is 147.8. The range for X generated by Flow\* using those settings is approximately 0 to 159. In HyCreate2 simulations for 1 sec from the corners of Rctr with numRange=7 using the max-thrust level-flight controller, the maximum value of X is 140.4.



**Figure F.4. Graph of X, Y from Reachability Computation from State S0**

**Benefit of Enforcing Invariants.** To evaluate the benefit of enforcing the flight safety invariants, we re-ran the experiment in the last row of Table F.2 without enforcing the flight safety invariants, and compared the resulting ranges for a few variables. Without the flight safety invariant to limit its value, the range of  $T$  increases linearly, reaching approximately  $-300..400$  after 1.02 sec; with the flight safety invariants, the range is  $T_{min}..T_{max} = 10..200$ . Similarly, without the flight safety invariant to limit its value, the range of  $L$  increases linearly, reaching approximately  $-2000..4000$  after 1.02 sec; with the flight safety invariants, the range is  $0..1500$ .

**Experiment with Region Rmax.** As a preliminary test of whether similar results will hold for other regions, we did a reachability computation for the region Rmax defined by the highest sub-range for each variable, using numRange=7, and with other Flow\* settings the same as in the last row in Table F.2. RENLE occurs after 0.83 sec of simulation time, with running time 17:14. This suggests that smaller regions will be needed near the edges of the initial guess for AdvancedMode.

#### **F.4.5 Feasibility Using HyCreate2**

All experiments in this section were performed by running HyCreate2.5 on uav.hyc2, except that HyCreate 2.6 on uav-HyCreate2\_6.hyc2 was used for one group of experiments as noted below. The .hyc2 files are available on VDL. HyCreate2.5 and HyCreate2.6 are available from Stanley Bak. The experiments were run on a desktop with an Intel Core 2 Quad @ 3.0GHz. HyCreate2's reachability computation is single-threaded, so the number of cores does not affect the running time for a single reachability computation

**Eliminating Dead Variables.** The non-deterministic model of the advanced controller does not use  $xT$  and  $xL$ , so it sets their derivatives to 0 (deterministically). We expected that this would ensure they always retain their initial values, hence they would not increase the number of states. However, we noticed towards the end of the experiments that the values of these variables vary. A preprocessor could be used to eliminate those variables from the HyCreate2 model when using the non-deterministic model of the advanced controller (as we do with Flow\*). When evaluating



the results for HyCreate2, keep in mind that the running time and memory usage could be reduced modestly by this straightforward optimization.

**Invariants.** As discussed above, we want to enforce several invariants (the inequalities corresponding to flight safety) during the reachability computation. Flow\* handles this nicely: the user specifies the invariants as inequalities between polynomial expressions over the state variables, and Flow\* drops states violating the invariant by computing the intersection of each computed Taylor model with the set of states satisfying the given inequalities. HyCreate2 allows the user to specify invariants, but in a different way. Specifically, the user provides a function, expressed using arbitrary Java code, which takes a box (i.e., a rectangular region of states) as argument and returns true if any state in the box satisfies the invariant, and returns false otherwise. If the function returns false, HyCreate2 discards the box; otherwise it keeps the entire box. Ideally, HyCreate2 would intersect the box with the invariant and discard parts of the box that do not contain states satisfying the invariant. To automate this in HyCreate2, a more tractable (than Java) representation of invariants would need to be adopted. An alternative approach is to allow the user to supply a Java function that performs the intersection and returns the result; this approach requires significantly fewer changes to HyCreate2 and imposes a fairly light burden on the user. In the meantime, we developed an approach that achieves part of this effect by changing the model instead of HyCreate2. We call the approach *enforcing invariants in derivatives*. Specifically, for each variable  $v$  constrained by an inequality in the invariant (namely, T, L, BA, V, and FPA), we modify the function that computes the derivative of  $v$  so that, if the value of  $v$  violates the inequality, then the function returns 0. This reduces the growth of the part of the box outside the region satisfying the invariant.<sup>7</sup> A side-effect of this change to the derivative function is that, even if the derivative was a monotonic function before this change, it is not monotonic after this change. In particular, for a box of which part satisfies the invariant and part does not, the extremal value of the derivative in the box might occur at a point where the box crosses the boundary defined by the invariant (e.g., if the invariant is  $v \geq c$ , the extremal value might occur at a point with  $v=c$ ). The function that returns the set of possible extremal points for the derivative of  $v$  should be modified accordingly. We started to implement this modification, but the resources were not available to debug and evaluate it. Consequently, this modification was not done for the experiments reported here, and the resulting regions of reachable states might be smaller than they should be. However, we expect that making this modification would have a modest effect on the set of reachable states and the running time, so our general conclusions should be unaffected. Furthermore, for the best HyCreate2 settings that we identified (small grid without large rectangle splitting), we also ran the reachability computation without enforcing invariants in derivatives (i.e., with the original definitions of the derivative functions, obtained by setting `nondetermObeyInvar=false` in `uav.hyc2`), thereby obtaining a sound result, with a moderately longer but still quite acceptable running time.

---

<sup>7</sup> We used a more proactive version of this technique to help prevent simulations from terminating early due to invariant violations, with partial success. Specifically, instead of waiting for the value of the variable to violate the inequality before setting the derivative to zero, we add the expected change during the current timestep (namely, the current value of the derivative times an estimate of the timestep duration) to the current value of the variable, and set the derivative to zero if the resulting value violates the inequality. However, invariant violations still cause early termination of simulations with the non-deterministic controller.

In the experiments in this section, we added an invariant limiting HA to  $-\pi.. \pi$ , because we are analyzing the system for at most 8 sec, and we know HA cannot reach  $-\pi$  or  $\pi$  in that amount of time (recall that HA0=0, and it takes about 30 sec for the UAS to turn around), so any states eliminated by this invariant are not actually reachable, even though they may appear to be reachable due to overapproximation.

All experiments in this section use an initial region corresponding to Rctr with numRange=3.

**Adjusting HyCreate2 Settings.** The main settings that control the speed and accuracy of HyCreate2's analysis are:

- Grid size for each variable. Recall that HyCreate2 represents regions in the state space as boxes, a.k.a. rectangles, whose edge lengths are controlled by the grid size.
- Face size ratio for each variable. This parameter controls the distance a face of a box can move in a single timestep. Larger ratios allow larger timesteps but reduce accuracy, because the maximum of the derivative over a larger region is used during face lifting.
- Regrid ratio for each variable. If large rectangle splitting is on, then after face lifting, boxes whose edge length exceeds the grid size times the regrid ratio are split into smaller boxes whose size is determined by the grid size.

For all experiments in this section, we used the same face size ratio for all variables, and the same regrid ratio for all variables.

**Small Grid and Rectangle Splitting.** We first tried a computation with a relatively small grid size <T: 1, L: 30, BA: 0.2, V: 4, FPA: 0.2, HA: 0.2, X: 10, Y: 10, H: 10, xT: 1, xL: 30>, and with the typical values of 0.1 for face size ratio and 1.7 for regrid ratio suggested in the HyCreate2 documentation. Large rectangle splitting was on. We terminated the computation after about an hour; it had reached only .0003 sec of simulation time.

We tried all three options for successor aggregation (namely, discretized aggregation, convex hull aggregation, and conditional discretized aggregation), but the results did not change significantly. We also tried to turn on pseudo-invariants with time step 0.1, but it seemed not to work: HyCreate2's output still said "Not using pseudo-invariants".

**Large Grid.** To reduce the number of rectangles, we significantly increased the grid size to <T:50, L:100, BA:0.4, V:50, FPA:0.2, HA:6, X:50, Y:50, H:50, xT:50, xL:100>, and we significantly increased the regrid ratio to 30. To increase the duration of each time step, we significantly increased the face size ratio to 1.0. Large rectangle splitting was on. The number of rectangles still "exploded", but progress was significantly faster than before. Pending rectangles jumped to about 140K early in the calculation, gradually decreased to about 43K by 0.5 sec of simulation time, and stayed there until 0.9292 sec of simulation time, when the JVM terminated with "java.lang.OutOfMemoryError: GC overhead limit exceeded" after about 130 minutes. The Java heap size was set to 3 GB. The output file contains 604K boxes. We speculate that this computation could run to completion (i.e., 1 sec of simulation time) with modestly more memory and running time; for argument's sake, say 150 minutes. Assuming the running time is similar for all 243 regions obtained with numRange=3, the end-to-end computing time for all the reachability computations on a cluster with 20 quad-core processors would be

approximately 7.5 hours. Some regions will probably take longer, so this should be considered an order-of-magnitude estimate.

**Accuracy with Large Grid.** To evaluate accuracy, we computed the smallest and largest values for each variable that appear in any box in HyCreate2's output.<sup>8</sup> The range is 0.5 to 260 for T, -96 to 3092 for L, 107 to 164 for V, and -202 to 194 for X. The ranges for T and L exceed the limits in the flight safety invariants (see Table F.1), so they provide no information. Based on simulations for 1 sec from the corners of Rctr with numRange=3 using the dive controller described in Section B.4.3, the maximum reachable velocity is 161.9. The upper bound of 164 for V from the reachability computation is only slightly higher than this, which is good, although if the reachability computation had continued to 1 sec of simulation time (recall that it ran out of memory at 0.9292 sec), it would have produced a slightly higher and hence slightly looser value. Based on simulations for 1 sec from the corners of Rctr with numRange=3 using the max-thrust level-flight controller described in Section B.4.3, the maximum reachable value of X is 154.8. The upper bound of 194 for X from the reachability computation is significantly higher than this.

**Large Grid and Large Regrid Ratio.** To further reduce the number of boxes, we further increased the regrid ratio. The explosion to 100K+ boxes still occurred for regrid ratio  $\leq 34$ . For regrid ratio 35, HyCreate2 generates about 20 rectangles and finishes in several seconds, because little regriding occurs.

**Accuracy with Large Grid and Large Regrid Ratio.** The computation with large grid and large regrid ratio has only moderately worse accuracy than the computation with large grid, despite its greatly reduced cost. With a regrid ratio of 35, the ranges defined by the smallest and largest values that appear in any box in the output are 8 to 282 for T, -57 to 3356 for L, 101 to 172 for V, and -213 to 193 for X. The ranges for T and L are not significantly looser than the ranges from the much more expensive calculation with regrid ratio 30, because the ranges from the more expensive calculation provide no information anyway. The upper bound for V is moderately looser than the upper bound from the more expensive calculation. The upper bounds for X are similar.

**Small Grid Without Large Rectangle Splitting.** Increasing the regrid ratio reduces regriding. The limiting case is to turn large rectangle splitting off. In this case, it should be feasible to use a small grid size; note that, with large rectangle splitting off, the grid size determines only the size of the initial grid. We turned large rectangle splitting off and used the small grid described above, with a face size ratio of 0.2. For this experiment, we used HyCreate 2.6 and selected the new Mfl ("Mixed face lifting") Reconstruct reachability method. Without enforcing invariants in derivatives, the computation reached 1 sec of simulation time in 50 sec of computation time; with enforcement of invariants in derivatives, the computation time was 34 sec. Figure F.5 and Figure F.6 contain graphs of X,Y and X, V, respectively. The simulations shown in the figure

---

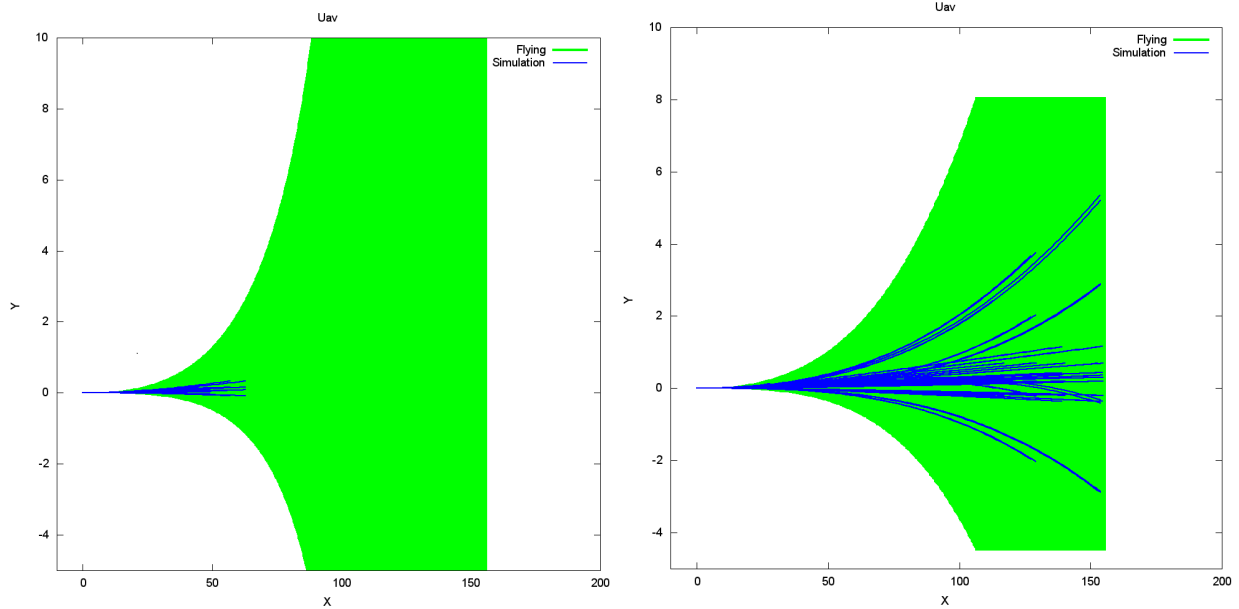
<sup>8</sup> Implementation detail: To do this, we used sed (a standard UNIX utility for non-interactive editing) to replace semicolons with commas in the output file result/reachability\_result.hrm, so the data is in csv format (we used sed because our interactive text editor choked on the 245MB file), opened the resulting file in Excel, and used formulas to compute the min of the lower bound column for each variable and the max of the upper bound column for each variable.

use a manually specified timestep of .01 sec. Comparing the results with and without enforcing invariants in derivatives, we see that this technique reduces the running time by about 30% and noticeably slows the spread in Y and V, although the benefits will be reduced when the necessary modifications to the extremal-point functions are incorporated. Without these modifications, the reachability computation misses some states: the maximum value of V in the top and bottom graphs in Figure F.6 are 160.1 and 163.6; the dive controller can reach 161.9.

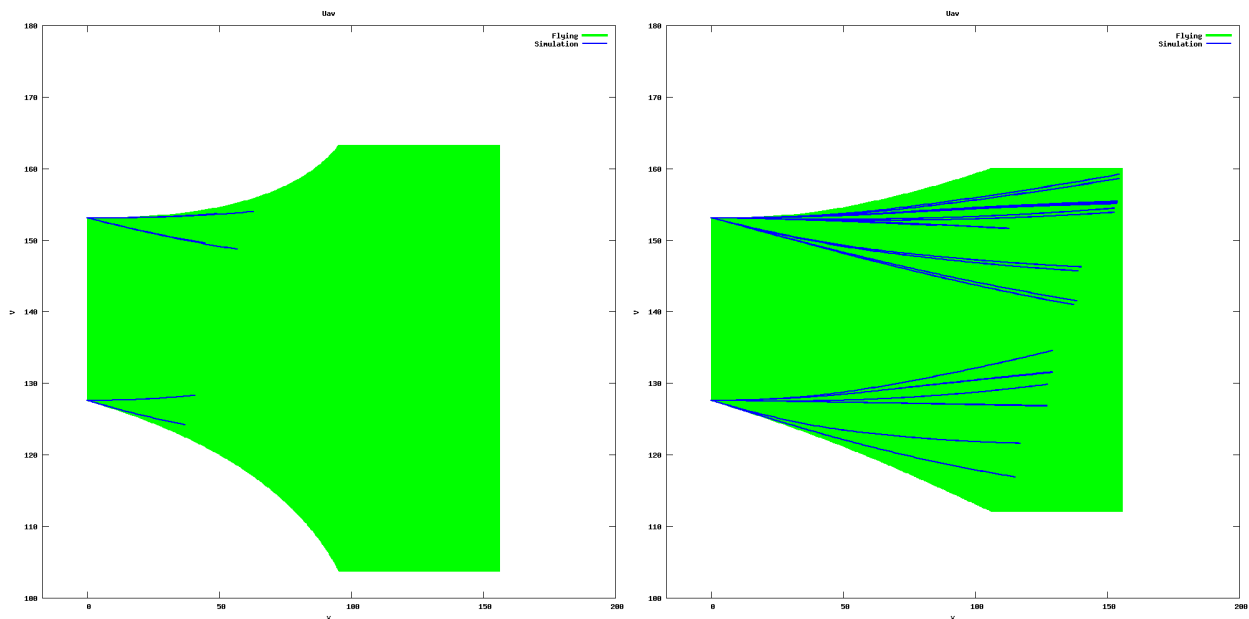
**Accuracy with Small Grid Without Large Rectangle Splitting.** The ranges defined by the smallest and largest values that appear in any box in the output are 9.9 to 200 for T, -0.2 to 1672 for L, 112 to 163.6 for V, and 0 to 155.7 for X. Based on simulations for 1 sec with a timestep of .01 from the corners of Rctr with numRange=3 using the dive controller described in Section B.4.3, the maximum reachable velocity is 161.9; a timestep of .001 gives the same result. The upper bound of 163.6 for V from the reachability computation is nearly tight. Based on simulations for 1 sec with a timestep of .01 from the corners of Rctr with numRange=3 using the max-thrust level-flight controller described in Section B.4.3, the maximum reachable value of X is 154.8. The upper bound of 155.7 for X from the reachability computation is nearly tight.

**Adjusting Size of Initial Region.** We ran similar experiments starting with an initial region containing only state S0 and got similar results, including the sudden change from a very large number of boxes to a very small number of boxes when increasing the regrid ratio. This indicates that experiments with larger values of numRange would produce similar results. Intuitively, HyCreate2 gets little benefit from smaller initial regions when using a relatively large grid size, because as soon as a regrid operation occurs, regions get expanded to the grid size.

**Benefit of Enforcing Invariants.** To evaluate the benefit of enforcing the flight safety invariants, we re-ran the experiment with regrid ratio 35 described above, except this time without enforcing the invariants. Instead of terminating with about 20 boxes, the computation generated 345K rectangles in the first step, after which we terminated it.



**Figure F.5. Graph of X, Y from Simulations and Reachability Calculation for Non-Deterministic Controller**



**Figure F.6. Graph of X, V from Simulations and Reachability Calculation for Non-Deterministic Controller**

## F.5 States Reachable With Safety Controller

This section describes the safety controller used to turn the UAS around as quickly as possible, and experiments to determine feasibility of reachability computations for 39 seconds of simulation time (the time needed for the UAS to turn around from some typical initial states). The answer might seem like a foregone conclusion, considering that accurate reachability

computations for 1 sec of simulation time for the UAS with the non-deterministic model of the advanced controller are pushing the limits of the tools. However, non-determinism significantly increases the size of the state space and hence the cost of the state-space exploration, so the answer is not so obvious. For example, although we saw in Section F.4.4 that Flow\* can analyze the UAS with the non-deterministic model of the advanced controller from initial state S0 for only 1.22 sec before RENLE, we will see that Flow\* can analyze the UAS with the safety controller from initial state S0 for 35.41 sec before RENLE.

These experiments have a similar design as the experiments with the non-deterministic model of the advanced controller in Section F.4. We experimented with reachability computations starting from the initial state S0 or the initial region Rctr defined in Section F.4.0.

### F.5.1 Summary of Conclusions for Reachability with Safety Controller

Flow\* is able to perform a reachability computation from initial state S0 for 35.41 sec of simulation time in about 22 min of running time. However, it is unclear whether Flow\*'s settings can be modified to extend this to 39 sec of simulation time. Furthermore, even if it is possible, reachability computations from larger initial regions, such as Rctr with numRange=7, are much slower; the reachability computation for that one region would be at least 8.5 hours. The end-to-end running time for all initial regions in the initial guess for AdvancedMode would be impractical, even on a small compute cluster.

HyCreate2 is able to simulate the system from individual initial states efficiently for 39 sec or more of simulation time, but reachability computations, even from a single initial state, make little progress, due to the large number of explored boxes.

HSolver (<http://hsolver.sourceforge.net/>) can be applied to this problem, but we did not try it, because, as mentioned in Section F.4.1, previous experience with HSolver showed that HSolver is much slower than HyCreate2 for hybrid systems like the UAS.

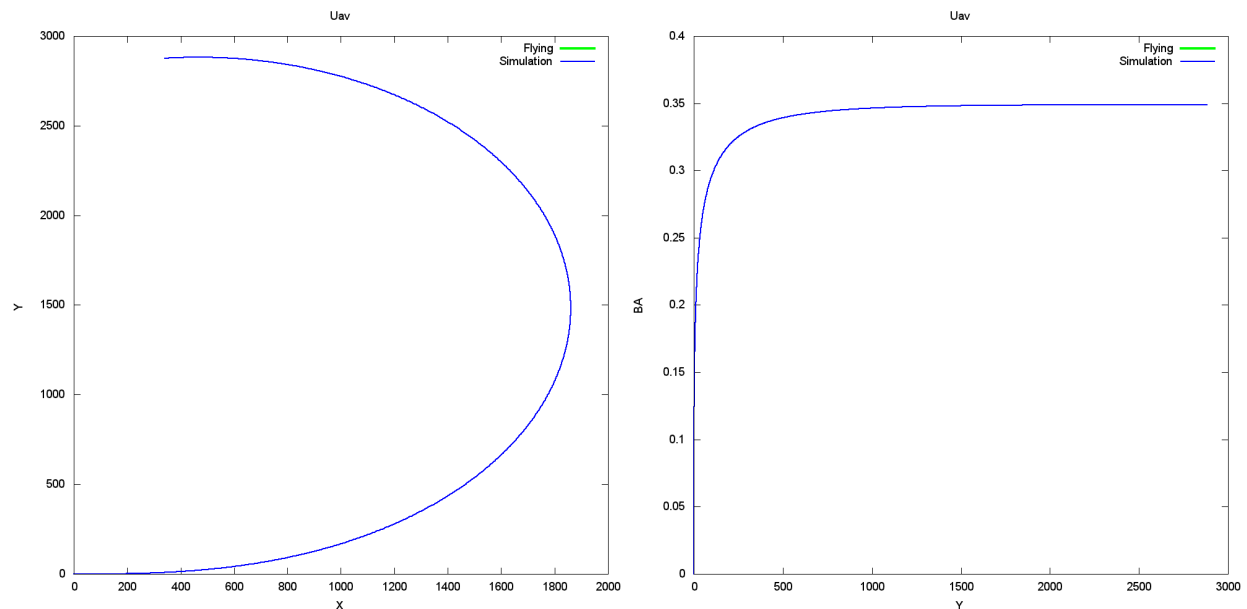
dReach can be applied to this problem, because the derivatives are deterministic. However, we do not expect dReach to be efficient for long-duration reachability computations, because the formula constructed by dReach (*cf.* the description of dReach in Section F.1.1) will be too large for efficient satisfiability checking.

### F.5.2 Design of the Safety Controller

Recall that the safety controller in this setting should turn the UAS around as quickly as possible, to avoid entering the no-fly zone. The safety controller used in these experiments is a variant of the controller in the MATLAB simulation of UAS guidance, which is a variant of the controller in the UAS guidance model presented in an earlier chapter of this report. The inputs to the controller are a commanded velocity Vcmd, a commanded height Hcmd, and a commanded bank angle BAcmd. The controller uses additional state variables xT and xL. T is controlled by a feedback loop; specifically,  $T' = p_T \cdot (T_{cmd} - T)$  where  $T_{cmd} = x_T + (K_{T1}/K_{T2}) \cdot x_{T'}$  and  $x_{T'} = (K_{T2} \cdot M_0 \cdot (V_{cmd} - V))$ . L is controlled by a feedback loop that brings V and H to commanded values Vcmd and Hcmd, respectively; specifically,  $L' = p_L \cdot (L_{cmd} - L)$  where  $L_{cmd} = x_L + (K_{L1}/K_{L2}) \cdot x_{L'}$  and  $x_{L'} = K_{L2} \cdot ((M_0 \cdot V_{cmd} \cdot ((K_{gc} \cdot (H_{cmd} - H)) - FPA)) + (K_{alt} \cdot (H_{cmd} - H)))$ . The values of the constants are not shown here but can be found in the MATLAB model, the

Flow\* model, or the HyCreate2 model. The controller in the UAS guidance model and the MATLAB simulation does not have a commanded bank angle as an input, because it is designed to head to a waypoint. For the safety controller, we adopted the following simple control law for the bank angle:  $BA' = KBA2 * (1 - BA/BA_{cmd})$ , where  $KBA2 = 0.1$ . We used the following commanded values:  $V_{cmd} = V_0$ ,  $H_{cmd} = H_0$ , and  $BA_{cmd} = BA_{max}$ . We did not experiment with other values of  $V_{cmd}$  and  $H_{cmd}$  to determine whether these values turn the UAS around as quickly as possible, but optimizing these values would not decrease the time needed to turn around enough to affect the conclusions below.

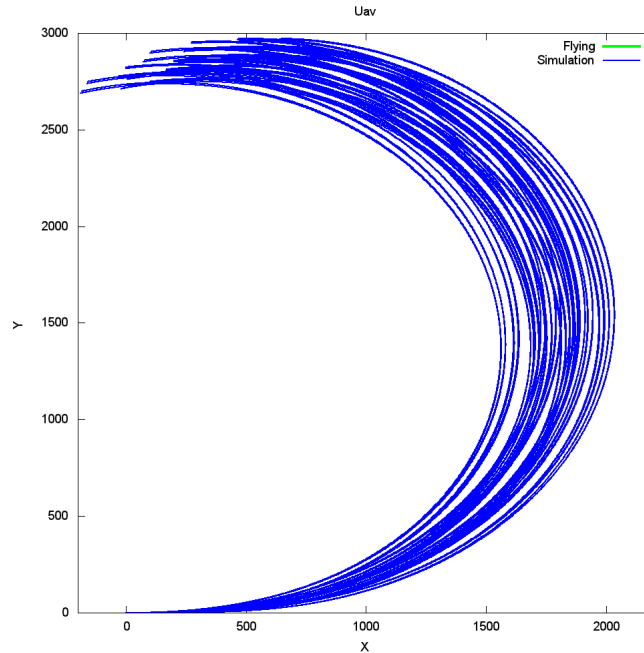
To evaluate this safety controller, we simulated the UAS with this controller, using HyCreate2. Consider a scenario starting from initial state  $S_0$  and with  $V_{cmd}=V_0$ ,  $H_{cmd}=H_0$ , and  $BA_{cmd}=BA_{max}$ . Figure F.7 contains graphs for 39 seconds of simulation time. The graph of  $X, Y$  shows that the UAS has turned around in that time. The graph of  $Y, BA$  shows that the controller smoothly brings  $BA$  from 0 to  $BA_{max}$  ( $Y$  is used on the horizontal axis as a surrogate for time, because time is an implicit variable in HyCreate2, so it cannot be used in graphs unless the model explicitly contains a variable equal to time;  $Y$  is a good surrogate for time, because it is a monotonic function of time in this scenario). The altitude gradually drops to about 960 feet and then stabilizes (we did not investigate whether the height would later increase back to  $H_0$ ). This could be mitigated by modifying the control law for lift to depend on the bank angle, but the loss of altitude is irrelevant for our purpose, so we did not experiment with such modifications.



**Figure F.7. Simulation of UAS with the Safety Controller**

### F.5.3 Simulations Used to Evaluate the Reachability Results

HyCreate2 was used to simulate the UAS with the safety controller for 39 sec starting from the corners of the region  $R_{ctr}$  defined in Section F.4.0, with  $numRange=3$ . HyCreate2 performs 256 simulations in less than a minute. Figure F.8 shows graphs of  $X, Y$  from the simulations. We see that, from every corner of this region, the UAS turns around within this time.

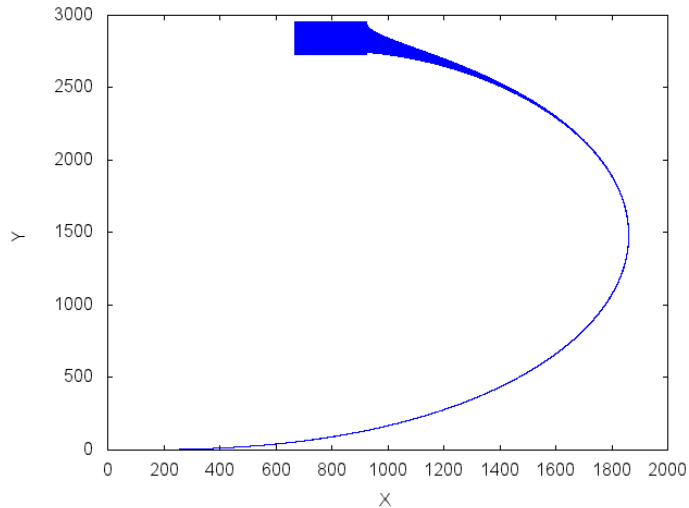


**Figure F.8. Graph of X,Y with Safety Controller from Corners of Rctr with numRange=3**

#### **F.5.4 Feasibility Using Flow\***

For these experiments, the Flow\* settings in the last row of Table F.2 were used. To determine whether a reachability computation for 39 seconds of simulation time is feasible at all, we first did a reachability computation starting from the single initial state S0. Figure F.9 contains a graph of X,Y from this computation. RENLE occurred at 35.41 seconds of simulation time, with running time of 22 minutes. Based on previous experience with Flow\* (including the experiments described in Section F.4), it seems unlikely that the RENLE can be delayed significantly by modifying Flow\*'s settings. Even if it is possible to modify the settings to postpone the RENLE to 39 sec for a single initial state, and even if this does not increase the running time significantly (which it probably would), the running time is likely to be impractical for larger initial regions. To estimate how much slower the reachability computation is for larger initial regions, we ran a reachability computation for the initial region Rctr with numRange=7, using the same Flow\* settings as above. We terminated it after 20 minutes of computation, by which time it reached only 1.53 sec of simulation time. This suggests that a reachability computation for 39 sec of simulation time from that initial region would take at least 8.5 hours, even if RENLE did not occur (and it certainly will occur).



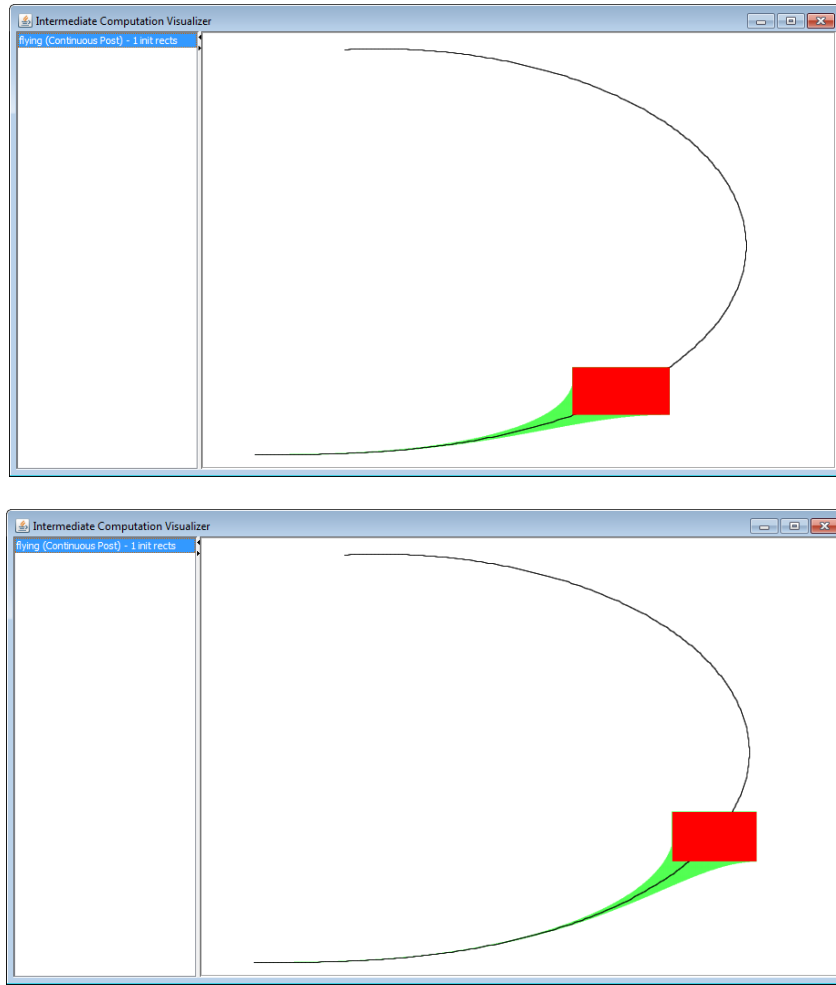


**Figure F.9. Graph of X, Y from Flow\* Reachability Computation**

### **F.5.5 Feasibility Using HyCreate2**

To determine whether a HyCreate2 reachability computation for 39 seconds of simulation time is feasible at all, we first did a reachability computation starting from the single initial state  $S_0$ , with large rectangle splitting off, with the large grid described in Section F.4.5, and with a face size ratio of 1.0 for all variables. The reachability computation quickly reached 2.38 sec of simulation time and then made no further progress before we terminated it after 44 minutes of running time.

The large grid size and large face size ratio may be causing too much accumulation of error, so we tried this with the small grid described in Section F.4.5 and a face size ratio of 0.1 (with large rectangle splitting still off). The computation reached 7.2472 sec of simulation time in about a minute and then made no further progress before we terminated it after 24 minutes of running time. We further decreased the grid size, multiplying each value by 0.1. The computation reached 11.7070 sec of simulation time in a few minutes and then made no further progress before we terminated it after 25 minutes of running time. The intermediate computation visualization, captured immediately before we terminated the computation, is shown in Figure F.10. We further decreased the grid size, multiplying each value by 0.1 again (hence 0.01 times the original small grid size). The computation was still making steady but slow progress, having reached 16.0849 sec of simulation time, when we terminated it after 106 minutes of running time. Regardless of whether this computation would also stop making progress at some point, the running time will be excessive, especially considering that this is starting from a single initial state, and an initial region like  $R_{ctr}$  with  $numRange=3$  will contain a very large number of rectangles when such a small grid is used.



**Figure F.10. Graph of X,Y from HyCreate2 Reachability Computation**

We also tried a reachability computation from  $S_0$  with large rectangle splitting on, using the same grid size and face size ratio, and using a regrid ratio of 35 for all variables. With the Java heap size set to 3 GB, the computation terminated with an OutOfMemory exception after several minutes.